AD-A054 508 HARRIS CORP MELBOURNE FLA ELECTRONIC SYSTEMS DIV F/G 9/2

INTELLIGENCE SECURITY SUBSYSTEM.(U)

MAR 78 F ANDERS, W MALL, R MCGILL F30602-76-C-0445

UNCLASSIFIED RADC-TR-78-33 NL

1 OF 4

AD

A054508

2

RADC-TR-78-33
Final Technical Report
March 1978

INTELLIGENCE SECURITY SUBSYSTEM

F. Anders
W. Mall
R. McGill
J. McLaughlin
B. Thompson

Harris Corporation

Approved for public release; distribution unlimited.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-78-33 has been reviewed and is approved for publication.

APPROVED: Fred Wilson
FRED WILSON
Project Engineer
Intelligence Systems Section

APPROVED: H Davis
HOWARD DAVIS
Technical Director
Intelligence & Reconnaissance Division

FOR THE COMMANDER: John P. Huss
JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (IRDA) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DDC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: RADC-TR-78-33

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle): INTELLIGENCE SECURITY SUBSYSTEM

5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report, Oct 76 - Oct 77

6. PERFORMING ORG. REPORT NUMBER: N/A

7. AUTHOR(s):
F. Anders
W. Mall
R. McGill
J. McLaughlin
B. Thompson

8. CONTRACT OR GRANT NUMBER(s): F30602-76-C-0445

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
Harris Corporation Electronic System Division
P.O. Box 37
Melbourne FL 32901

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
62702F
45941022

11. CONTROLLING OFFICE NAME AND ADDRESS:
Rome Air Development Center (IRDA)
Griffiss AFB NY 13441

12. REPORT DATE: Mar[illegible]978

13. NUMBER OF PAGES: 330

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same

325 p

15. SECURITY CLASS. (of this report): UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report):
Approved for publication; distribution unlimited.

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report): Same

18. SUPPLEMENTARY NOTES:
RADC Project Engineer: Fred Wilson (IRDA)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number):
Computer Security
Encyphered Data Base
Security of Data Base

ABSTRACT

20. ABSTRACT (Continue on reverse side if necessary and identify by block number):

The report finds that it is technically feasible to achieve security in computer systems and specifies an engineering model that would confirm the findings.

ABSTRACT

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

408 972

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

TABLE OF CONTENTS

			Page
Paragraph	1.0	INTRODUCTION	2
	1.1	The Problem of Data Protection	2
	1.2	Scope	3
	1.3	Method	4
	1.4	Tactics of Computer Criminals	4
	1.5	Evaluation of Candidate Architectures	6
	1.6	Conclusions and Recommendations	9
	1.7	Compliance With the Statement of Work	10
	2.0	AN/GYQ-21(V) BACKGROUND	14
	2.1	PDP-11/45	14
	2.2	RSX-11D	15
	2.2.1	Functions	16
	2.2.2	References	16
	2.2.3	The RSX-11D Executive	17
	2.2.4	Input/Output Operations	17
	2.3	IAS Application at HQ SAC, ADCOM and the DIA	17
	2.3.1	Intelligence Networks	17
	2.3.2	Stand-Alone, Data Base Systems - NMIC	19
	2.3.3	Communications Processor for a Large Data Base System	22
	2.4	Typical Configuration of the IAS	27
	3.0	THREAT	30
	3.1	Criminal Tactics Given Command of the System	30
	3.2	Gaining Command by Exploiting a Trapdoor	32
	3.2.1	Example of a Trapdoor in MULTICS	32
	3.2.2	Data Base Query Trapdoor	33
	3.2.3	Ease of Inserting and the Difficulty of Finding Trapdoors	34
	3.3	Gaining Command of the System by Exploiting Hardware Flaws	34
	3.4	Exploiting Software Flaws	35
	3.4.1	Insufficient Argument Validation	35
	3.4.2	Master Mode Transfer	36
	3.4.3	Unlocked Stack Base	38
	3.5	Penetration of RSX-11D	39
	3.6	Conclustions on the Threat	44
	4.0	EVALUATION OF CANDIDATE APPROACHES	46
	4.1	Existing IAS	46
	4.2	Entrance Guards	47
	4.3	Protected Access Controls	48
	4.4	Security Monitors	50
	4.5	Isolated Security Monitors	50
	4.6	Data Base Guard	51
	4.7	Tag Approach	53
	4.8	Enciphered Records	54

TABLE OF CONTENTS (Continued)

			Page
Paragraph	4.9	Other Approaches	56
	4.10	Preliminary Findings	57
	5.0	AN EXPERIMENTAL, ENCIPHERED DATA BASE SYSTEM	60
	5.1	Background	60
	5.1.1	Objectives	60
	5.1.2	Method	60
	5.1.3	Message Flow in the Model	61
	5.1.4	Problems Examined by the Feasibility Model	61
	5.1.5	Summary	63
	5.2	LSI-11 Top Level Flow	63
	5.2.1	Exit Guard Design Approach	63
	5.2.2	LSI-11 Top Level Flow	65
	5.2.3	Initialization of the LSI-11	65
	5.2.4	Read Keyboard	66
	5.2.5	Process Entry	67
	5.2.6	Output to the 11/45	68
	5.2.7	Input Results	68
	5.2.8	Verify Tag	69
	5.2.9	Good Tag	70
	5.2.10	Bad Tag	70
	5.3	PDP-11/45 Top Level Flow	71
	5.3.1	Top Level 11/45 Flow	71
	5.3.2	Initialize 11/45	71
	5.3.3	Input Messages	72
	5.3.4	Message Source	73
	5.3.5	Console One	73
	5.3.6	Console Two, Three, Card Reader	74
	5.3.7	Unknown Source	74
	5.4	Query Processing	75
	5.4.1	Top Level Flow	75
	5.4.2	Accept Request	77
	5.4.3	Generate Intermediate Language	77
	5.4.4	Errors	78
	5.4.5	Send Administrative Error Messages	78
	5.4.6	Execute Intermediate Code	79
	5.4.7	Send Results to Console	79
	5.5	Data Base Query Operations	81
	5.5.1	Storage and Retrieval Block Diagram	81
	5.5.2	Terminal Input/Output Block Diagram	81
	5.5.3	List of Statements	81
	5.5.4	The Display Statement	81
	5.5.5	The Print Statement	84
	5.5.6	The Delete Statement	84
	5.5.7	The Add Statement	84
	5.5.8	The Change Statement	85
	5.5.9	The If Statement	85

TABLE OF CONTENTS (Continued)

			Page
Paragraph	5.6	Simplified Record Tagging and Detagging	85
	5.6.1	Key Initialization	86
	5.6.2	Tagging Process	86
	5.6.3	Typical Record Tag	87
	5.6.4	Detagging Process	87
	5.7	Detailed Flow Diagrams - Query Processing ..	87
	5.7.1	Accept Request	87
	5.7.2	Generate Intermediate Language	92
	5.7.3	Code Execution	101
	5.8	Detailed Flow Diagrams - 11/45	106
	5.9	Detailed Flow Diagram - LSI-11	106
	5.10	Overall Operation of ISS System	112
	6.0	RED-BLACK MULTIPROCESSING	122
	6.1	Introduction	122
	6.2	Threat Environment and Related Assumptions	122
	6.2.1	Personnel Assumptions	122
	6.2.2	Equipment Assumptions	125
	6.3	System Design Goals	125
	6.4	System Design Description	126
	6.5	Security Evaluation	127
	6.6	Performance/Cost Evaluation	128
	6.6.1	Multiprocessor Versus Uniprocessor System Cost	130
	6.6.2	Special Security Features Cost	130
	6.7	Summary and Conclusions	130
	7.0	DATA BASE GUARD APPROACH	133
	7.1	A Protection Scheme	133
	7.2	The Data Base: Files and Records	133
	7.2.1	Writing the Data Base	136
	7.2.2	Access Speed	136
	7.2.3	Design	137
	7.2.4	The CIU Bus	137
	7.3	The CIU Software Interface	137
	7.4	THE CIU Hardware	142
	7.5	The Controller	142
	7.5.1	The Microprocessor	142
	7.5.2	Read Circuitry	152
	7.5.3	The Write Circuitry	152
	7.6	Conclusion	152
	8.0	FINDINGS, CONCLUSIONS AND RECOMMENDATIONS	158
	9.0	SPECIFICATION OF A SECURE DATA BASE SYSTEM	162
	9.1	Scope	162
	9.1.1	Identification	162
	9.1.2	Functional Summary	162
	9.2	Applicable Documents	162
	9.3	Requirements	162

TABLE OF CONTENTS (Continued)

			Page
Paragraph	9.3.1	Program Definition	163
	9.3.2	Detailed Functional Requirements	169
	9.3.3	Adaptation	169
	9.4	Quality Assurance Provisions	169
	9.4.1	Introduction	169
	9.4.2	Test Requirements	169
	9.4.3	Acceptance Test Requirements	169
	9.5	Preparation For Delivery	169

EVALUATION

This final technical report presents a summary of work performed by Harris Corporation Electronic Systems Division under Contract F30602-76-C-0445. The objectives of this effort was to develop a security monitoring subsystem for the AN/GYQ-21(V) Interactive Analysis System (IAS). The approach of this initial study effort was to evaluate technical feasibility and develop detailed specifications for prototype fabrication. The significance of this effort in relation to Air Force technical objectives is that the report finds that with recent advancements in microcomputer technology it is technically feasible to achieve security in computer systems and recommends an engineering model that would confirm the findings.

Fred Wilson

FRED WILSON
Project Engineer

1.0 SUMMARY

1.1 Purpose. The purpose of the "Intelligence Security Subsystem" study was to find ways of protecting data in the Interactive Analyst Console, the AN/GYQ-21(V).

1.2 Technical Problem. Access to classified documents is controlled by a system of locks, safes, vaults, protective facilities, cleared employees and security personnel. Access controls to protect the data stored in computers are complex hardware/software mechanisms. Vulnerability tests described in Section 3 of the report have proven these controls to be ineffective. The controlled data, function, equipment, and uses differ considerably in application of automated intelligence systems as used by the Defense Intelligence Agency, the Unified and Specified Commands, and the Services. Section 2 of the report describes some of the applications and presents the conclusion that a simple model incorporates the essential features of access control problems in most applications.

In this model the data requiring data control access resides on disk file and is organized in the form of records, messages, reports, journals and other forms. Data is assumed to be organized into variable length records for simplicity. Each record is assigned a security level, intelligence compartment and "need to know" category. The term "intelligence compartment" is defined as the privileges required to access a record. It is convenient, though inaccurate, to think of the data base as comprising a number of mutually exclusive compartments, each containing a number of records with the same privileges required of a user to access those records - much of the library might be separated into isolated compartments, each containing a number of books. Essentially, the user control problem is to allow access to all authorized compartments and to prohibit access to all other compartments.

1.3 General Methodology. The ideal method of estimating protection afforded by a particular computer architecture would be to build a system and invite the world's leading computer crime experts to try and steal data. The time it would take to accomplish the espionage would be a measure of the protection afforded by that computer.

Obviously, such a method would be both time consuming and expensive. Performance requirements would set the minimum penetration time at somewhere in the range of a decade to a century. Exhaustively testing the hypothesis the minimum penetration time of 10 years, would be out of the question. Still, penetration teams provide a valuable service. Their tactics must be understood before computer systems can be designed, much as safe-cracking tactics must be understood by vault designers.

The method of evaluating alternative computer architecture used in this study is to assume the criminal has gained access to the terminal, and is both highly trained and talented in employing the most effective

penetration tactics. The figure or method for a particular architecture is the estimated penetration time. Comparisons of alternative computer architectures is then accomplished by examining the penetration time estimates.

1.4 Technical Requirements. The specification for a secure data base system is contained in Section 9 of the report. It calls for the design of a secure data base system prototype utilizing enciphering techniques to control access to the data.

2.0 CONCLUSION

The results of the study demonstrate that it is feasible to protect data. The most effective means is to encipher the data base and control access by exit guards at the terminal.

- Database Guard - The Database Guard approach allocates one central processing unit for each security compartment. Upon completion of the sign-in procedure, the user is switched to the computer that corresponds to the user's access privilege. A Secret privilege user will be switched to a Secret computer and will be denied access to Top Secret records in this computer. Potential vulnerabilities are the database guard (multiported disk controller) and the initial switching mechanism. Design of the database guard is presented in Section 7. It was shown that the microprocessor that controls communication into and out of the database has a small program that can be certified. This will prevent the insertion of trapdoors that would defeat the protection mechanism. The initial switching mechanism transfers the Secret user to the Secret IAS. It also represents a potential vulnerability. If the mechanism can be tricked into switching the Secret user to the Top Secret machine, the protection mechanism can be defeated. Fortunately, the microprocessor that performs the initial switching function has a small program that can be certified. We conclude that the potential vulnerabilities can be made impenetrable and as a result, the data base guard approach can be used as a basis to develop a secure IAS.
- Enciphered Record - The Enciphered Record approach presumes the processor is a hostile environment and applies the principles of RED/BLACK isolation the same as used in telecommunications. The records of the data base are enciphered, and even if accessed are useless without the ability to decipher them. The approach has appeal in that it is an elegant engineering solution to the problem of protecting data. Potential vulnerabilities of the system consist of: the access control mechanism, RED processor, cracking the code, and stealing the key. The access control mechanism, like that of the data base guard, is comprised of a small microprocessor with certifiable software. Alternatively, the whole mechanism may be hardwired, thereby, eliminating the software problem. The problem of isolating the RED processor was addressed in Section 7 with the conclusion that the BLACK processor can be totally isolated from the RED processor. We have assumed throughout that a block enciphering algorithm will be used that is impenetrable. For the initial design purposes, the NBS algorithm will be used, although it is recognized that this is unacceptable for military applications. We assume the Government will supply a military-acceptable block encryption algorithm. The final way the system is defeated

is by stealing the key. This can be made impossible by a number of means. One way is to manually insert the key at the guard and design the guard in such a way that if a penetration is attempted the guard will destroy the key before anyone can gain access to it. The sections on RED/BLACK multiprocessing and the enciphered data base system were intended to demonstrate the complexity of making this approach invulnerable. On the basis of information set forth in the report we conclude that the potential vulnerabilities of the enciphered record approach can be made impenetrable and, therefore, the enciphered record can be used as a basis for developing an impenetrable IAS.

- Tag Approach - The Tag Approach to record security utilizes an enciphered tag to inform the exit guard of record classification. The approach is the simplest of the three and only requires an access control mechanism. Since the records are in the clear, RED/BLACK multiprocessing is not required. The approach is an effective measure against the threat of accidental disclosure. For this threat, it is the cheapest and will do the job. We, therefore, conclude that this approach is optimum against a benign threat. On the ability to withstand the attack by computer criminals, we are much less confident. As in the case of the enciphered record, potential vulnerabilities are: access control mechanism, cracking the code, and stealing the key. But inlike the enciphered record approach with the data and records in the clear, the tag approach utilizes records that are in the clear. Therefore, they can be copied and dumped on a terminal or disguised as an error message and sent to the user. There are countermeasures to this vulnerability such as placing an exit guard at every terminal device and designing nonrecord communication containing an unforgeable tag, that describes the transaction as such. In practice, these may or may not be easy to implement. There is some doubt and, therefore, we do not recommend a tag approach against the threat of malicious attack at this time.

3.0 RECOMMENDATIONS

Since, of the three possible approaches, one is at least questionable in its ability to withstand criminal attack, there remains only two possible alternatives for recommendation in its use in development of an engineering model. The data base guard approach appears the least vulnerable, but it is the most expensive. We, therefore, do not recommend it for implementation. There is a second reason for this. There is very little question of feasibility and building an engineering model of the data base guard approach would prove very little. Accordingly, we recommend that an engineering model of the enciphered record approach be developed as specified in Section 9 of the report. Some experts will undoubtedly argue that building an admittedly special purpose data base system is not as productive an area of research as approaches to general purpose solutions, such as on-going efforts to develop secure operating systems. Harris disagrees. First of all, a secure data base system would satisfy many military needs, not only in the intelligence community but also in command and control. Secondly, building special purpose machinery is an inductive approach to the general purpose problem. We have already observed that RED/BLACK multiprocessing is necessary to solve the secure data base problem. Quite possibly, a more general multiprocessing architecture will provide a basis for a multilevel secure computing system.

4.0 IMPLICATIONS FOR FURTHER RESEARCH

Harris Electronic Systems Division recommends that the Government continue the research by implementing the prototype as specified in Section 9 of the report.

SECTION 1.0

INTRODUCTION

1.0 INTRODUCTION

Electronics has made advances in the collection, distribution, and production of intelligence possible. Recently, the availability of data base teleprocessing systems permitted automation of certain intelligence functions with attendant advantages of rapidly correlating information.

Unfortunately, computers suffer from a serious performance deficiency - they cannot protect data. This flaw is fatal in intelligence applications where no threat of espionage can be tolerated.

Despite the inability of computing machines to control data sharing, certain restricted applications are being developed. The restriction pertains to the users in that each (user) must be cleared to receive any and all information from the data base. Not only must the user have security clearances corresponding to the highest level of data contained in the computer, but must also be allowed access to all intelligence compartments.

In practice, this restriction limits the exploitation of automated data base systems. The proliferation of "high water" systems would destroy the information protection afforded by compartmentalization.

The purpose of this study is to find ways to improve the protection of data in the Interactive Analysis System, AN/GYQ-21(V), which is used extensively to automate intelligence and command control functions.

1.1 *The Problem of Data Protection.* Access to classified documents is controlled by a system of locks, safes, vaults, protected facilities, as well as cleared employees and security personnel. Access controls to protect data stored in computers are complex hardware/software mechanisms. Vulnerability tests, described in Chapter 3, have proven these controls to be ineffective.

The controlled data, function, equipment, and uses differ considerably in the application of automated intelligence systems as used by the Defense Intelligence Agency, the Unified and Specified Commands, and the services. Chapter 2 describes some of these applications and presents the conclusion that a simple model incorporates the essential features of the access control problems in most applications.

In this model, the data requiring controlled access resides on a disk file and is organized in the form of records, messages, reports, journals, and other forms. Data is assumed to be organized into variable length records for simplicity. Each record is assigned a security level, intelligence compartment, and "need to know" category. he term "intelligence compartment" is defined as the privilege required to access a record. It is convenient, though inaccurate, to think of the data base as comprising a number of mutually exclusive compartments, each containing a number of records with the same privilege required of a user to access

those records - much as a library might be separated into isolated compartments, each containing a number of books.

Essentially, the user control problem is to allow access to all authorized compartments and prohibit access to all other compartments.

1.2 Scope. An electronic record access control system can be defeated in a number of ways. Most are considered in Chapter 3, "Threat Analysis." However, three threats that have well understood countermeasures utilizing technology significantly different from that considered in this report are specifically excluded from consideration in this study. These threats are:

- Masquerading - Each user must identify himself when signing on the computer. Next, the user must validate identity, usually by typing in a password. This method of identity verification is vulnerable. Improved methods based upon fingerprints, voice prints, hand geometry, signature, and others are being explored; not only to improve computer security, but for commercial, industrial, and residential security applications as well.
- Wiretapping - A user with low access privilege may tap lines of a user with higher access privilege thereby gaining unauthorized access. This can be, and is frequently, prevented by enciphering data or by security guard surveillance.
- Monitoring emanations - Data communication and processing within the computer results in emanations which can be monitored by instruments. Vaults, shields, isolation components, and other means of control are commonly employed.

In addition to the three threats, one approach is also excluded from consideration.

- Secure software - The problem of access control would be solved, by definition, if secure software were available. Specifically, efforts have been made to develop a kernel for the PDP-11/45 which could be adopted for compatibility with RSX-11D utilized in the AN/GYQ-21(V). Efforts to develop secure software, unfortunately, have not succeeded to date. For example: the Air Force Electronic Systems Division abandoned efforts in this area after 6 years and several millions of dollars in expenditures. The bibliography of reports issued upon termination of similar activities identifies 76 studies. The concensus is that this study is unlikely to succeed where those had failed. IBM announced a $40 million program on computer security some years ago. Their current view[1] is that development of a secure operating system is beyond the state of the art.

---

[1] As expressed by Mr. Courtney, IBM Manager of Computer Security at the annual Computer Security Institute meeting in New York City, November 1976.

The decision to exclude secure software from consideration carries with it the implied decision to abandon internal access control mechanisms as these are predicted upon software. Accordingly, the scope of solutions considered is confined to external access control mechanisms. This approach is believed to be a fruitful field of exploration because of the rapid technological advance as evidenced by the microprocessor, enciphering devices on a chip and the rapid and general decline of hardware and firmware prices.

1.3 Method. The ideal method of estimating the protection afforded by a particular computer architecture would be to build the system and invite the world's leading computer crime experts to try and steal data. The time it would take to accomplish the espionage would be a measure of the protection afforded by that computer.

Obviously, such a method would be both time consuming and expensive. Performance requirements would set the minimum penetration time at somewhere in the range of a decade to a century. Exhaustively testing the hypothesis that the minimum penetration time is 10 years, would be out of the question. Still, penetration teams provide a valuable service. Their tactics must be understood before computer systems can be designed, much as safecracking tactics must be understood by vault designers.

The method of evaluating alternative computer architectures used in this study is to assume the criminal has gained access to a terminal and is both highly trained and talented in employing the most effective penetration tactics. The figure of merit for a particular architecture is the estimated penetration time. Comparison of alternative computer architectures is then accomplished by examining the penetration time estimates.

1.4 Tactics of Computer Criminals. Recorded and postulated criminal tactics are many and varied, involving attacks by maintenance, operations, and development personnel (alone and in conspiracy). In essence, all espionage attacks proceed in two steps: first, the criminal gains command of the system; and then dumps the protected data.

To understand the specifics of the tactics, suppose a criminal somehow gained the privilege of reading or writing any word in the computer's main memory. The criminal could then proceed with any or all of the following tactics:

- Dumping the password file - This tactic may, and frequently is, countered by enciphering the passwords. Still, a criminal can get all users' passwords if clear text copies can be obtained. These passwords can then be used to masquerade as a user with higher privileges.
- Falsifying the access control list - When a new user signs on a typical system, the user's Social Security number must be supplied as a means of identification (ID). The computer checks the Social Security number and finds the access privileges for that particular

user. (The list of users and their access privileges are contained in the access control list.) If a malicious user is free to read and write information on core storage, access privileges can be falsified.

- Masquerading as a user with higher privilege - A related technique falsifies an ID held by the computer system to identify the current user in a time-sharing system. For example: if the ID is falsified, the malicious user can masquerade as the system administrator with access privileges to all parts of the system.
- Commanding the operating system to dump privileged data - If the user is allowed to read and write into any access location, it is possible to modify and, therefore, command operating system modules. All the operating systems - RSX-11D, GECOS III, etc., - are in the public domain. Copies of the listings can be readily obtained. An accomplished criminal can find the location of the operating systems in core memory and modify them as desired. For example: a data base retrieval module can be modified to dump any records contained in the data base. While there may be some difficulties encountered in reading the octal codes and in locating the modules in memory, these problems are not insurmountable and could certainly be accomplished within a few days.
- Writing and executing programs that dump any information in the data base - It has been sometimes naively assumed that, by eliminating the programming capability of the user or by eliminating or protecting the operating system, a penetrator could not gain unauthorized access. Such is not the case. Even if the system has no operating system modules to command, the user can create such modules by programming in machine language. Such an approach is awkward and more difficult than programming in assembly or higher level language, but it is certainly well within the realm of feasibility.

The preceding list is by no means complete, but it will serve to illustrate tactics that can be effective for an espionage agent.

The criminal can also crash the system by writing into the operating system and by other means. Such a sabotage attack is not, in our view, considered serious because service would only be denied for a brief period of time. For example: in the National Military Intelligence Center (NMIC), copies of the operating system are available on-line and can be reloaded in a matter of minutes. The criminal might persist by attempting to crash the system several times which would be extremely dangerous for him. For this reason, the study does not consider the problem of preventing a sabotage attack. Indeed, this is thought to be improbable.

The conclusion is inescapable - if there is any way the criminal can gain the power to read and write any one word in main memory, the penetration time to unauthorized data will be minutes, or even seconds; rather than decades as prescribed by required performance.

Chapter 3 provides details on how the criminal can gain the ability to read or write any word in computer main memory. The category of tactics is summarized as follows:

- Trap door attack - The simplest way for the malicious user to gain control over the system is by use of what's termed a "trap door" that can be inserted by an accomplice in minutes. The trap door may be as small as one or two instructions and is almost impossible to detect. It is like looking for a needle in a haystack to find two instructions in tens of thousands.
- Exploiting software flaws - While the insertion of a trap door is a rather simple task, the RSX-11D operating system is developed without security controls and maintenance programs are communicated in an unclassified form. Even so, it is still unnecessary for the penetrator to rely on trap doors. A flaw in the software will give the criminal the needed capability to read and write an arbitrary word in main memory. Examples are detailed in the body of this report.
- Exploiting hardware flaws - That numerous software flaws exist in RSX-11D is certainly the case. One example is given in the body of the text. Even if this is not the case, there are probably hardware flaws in the PDP 11/45 that could be exploited. Examples of how such a hardware flaw was found in the MULTICS system are given in the body of the report.

1.5 *Evaluation of Candidate Architectures*. The analysis documented in Chapter 3 estimates the time it would take a competent criminal to gain unauthorized access to one or more records stored in the data base. Penetration time estimates are compared for a number of candidate protection methods. Comparison of alternative approaches, i.e., architectures, is based upon two criteria. First, espionage must be impossible for all practical purposes, i.e., penetration time estimates must be years, if not decades, to access a single record. Second, the solution should be affordable, i.e., the total incremental cost including hardware and software modifications should be less than the cost of alternatives. The engineering problem is one of selecting an approach that has an acceptable performance (penetration time) and a minimum cost.

The variation in cost and performance among candidate computer architectures is so pronounced that sophisticated analysis is not required in most cases. Seven of the ten candidates examined totally failed to prevent espionage. Of the remaining three, a serious question about the performance of one candidate exists. Deciding between the remaining two requires detailed preliminary design.

Highlights of the Chapter 4 analysis are summarized below.

- Existing AN/GYQ-21(V) - Harris ESD penetrated the IAS[2] in 2 hours in an experiment conducted as part of the study. Undoubtedly, many exploitable flaws exist in both RSX-11D, the application software, and the PDP-11 hardware. No attempt at identification was made since their existence is academic. A trap door that would allow any terminal user total access to the system in a matter of minutes could easily be inserted.

  No consideration should be given to compartmentalize security using the IAS. Denying programming access to terminal users is an ineffective procedure since the trap door provides programming access, although only in machine language.
- Security monitors - It is considered conventional wisdom to monitor interactive dialogue for the purpose of detecting computer crime, much as some banks continuously photograph their lobbies to obtain evidence when a crime occurs. The analogy is a bad one because espionage can be completed without producing a detectable audit trail.

  For example: A terminal user could type in a character sequence such as "ZYQMG" that would trigger a trap door to dump certain unauthorized information at the penetrator's terminal. Logs of the traffic both into and out of the data base would provide no clue that unauthorized data had been requested or that the request had been honored.

  Bypassing security monitors embedded in existing systems is even simpler. Even if incriminating information is collected, the espionage agent can simply write and read the logs so as to erase any incriminating information. In one way, security monitors are worse than useless because they seem to provide a measure of insurance, yet are totally ineffective against a competent computer criminal.
- Isolated security monitors - It is entirely possible to build microprocessors or minicomputers that are totally divorced from the main computing system and record interactive traffic. Logging functions are relatively simple, so the software can be held to a minimum. This can be certified, and assurances against the penetrator falsifying the logs can be made. Still, this approach cannot deal with the problem of interpreting stimulus and response, cited above. Accordingly, we also view isolated security monitors as ineffective with one important exception: While security monitors by themselves will not preclude the possibility of espionage, they can be an effective aid when used in conjunction with other methods.

---

[2] Interactive Analysis Station (IAS) is another name for the AN/GYQ-21(V).

- Entrance guards - As the name implies, the concept involves placing automated devices to monitor the user input portion of the dialogue. Unlike security monitors, entrance guards operate in real time. When authorized access is detected, the Access Control Center (ACC) is notified. The security officer, resident at the ACC, is notified of the anomaly and takes preventive action. Entrance guards have the same fatal flaws of isolated security monitors - They cannot detect attempted espionage by an accomplished computer criminal with 100 percent accuracy.
- Exit guards controlling access to enciphered records - This approach requires a radically different architecture. The data base system comprises a number of variable length records that are all enciphered. Assuming that the espionage agent can freely access the records and that the information is protected by the enciphering process, theory is that the criminal cannot crack the code in any reasonable period of time without a code key. Harris ESD believes such an approach has merit because it is capable of satisfying the penetration time requirements. In addition, this architecture does not appear to have a serious cost penalty because of inexpensive hardware availability for encryption and decryption.

  Feasibility is a far more serious question. At some point in the computer, operations must be performed on the records in all the data base applications that Harris ESD studied. For example: In the National Military Intelligence Centers, incoming messages are read by the computer to determine distribution of message copies. Obviously, no reading can take place if the records are enciphered. The problem can be circumvented by having two processors totally isolated from each other. The first, called the "black processor," always contains enciphered records. It is assumed that the criminal can freely access any of those records, but is denied the information content due to lack of an enciphering key. The second processor, called the "red processor," receives records from the first processor; and deciphers, processes, reenciphers, and returns the messages to the black processor. RED-BLACK isolation is adhered to in the two processors much as they are in standard communications equipment.

  The problem with this approach is that RED-BLACK isolation must be achieved. The danger is that a secure multiprocessing system (RED-BLACK processors) is similar to the isolation of multiprogramming capability which has defied solution for the past decade. This turns out not to be the case. Results of analyses in Chapter 6 are positive that RED-BLACK isolation can be achieved.
- Exit guards operating on control tags - The problem of secure multiprocessing can be bypassed if, instead of enciphering the entire message, only a tag is enciphered. The tag contains a nonforgeable compartment identification. The methods of preventing forgery are critical to the estimates of penetration time presented in Chapter 4. In summary, it is felt that the time to forge a tag can be maintained to an arbitrary long period provided a sufficiently sophisticated algorithm for enciphering the record and computing parity bits is maintained.

The control tag approach requires additional equipment. An exit guard must be placed between each terminal in the system, or one guard must be multiplexed among the number of terminals. An access control center must be available to identify the user and pass access rights to the exit guard. Further, all incoming messages must be tagged in order for the system to work effectively. Preliminary cost estimates indicate that this would amount to an incremental cost of perhaps 1 to 3 percent of the hardware and software costs of a large data base system. Harris ESD does not feel such a cost prohibitive, but no general statements should be made. Judgments should be made on an application-by-application basis.

A severe problem with the tag approach is that of preventing a malicious user from bypassing the exit guard. Results of analysis are inconclusive on this point.

- Data base guards - A positive method of controlling access to the data base is to allocate one IAS for each security compartment and control access from the data base to that computer through a multiported memory. In that way, the compartment identification contained in each record is examined by the multiported memory after the request has been honored. If the addressed computer does not have the required access rights, the multiported controller denies the request and notifies the access control center. While this architecture appears to be unpenetrable in any period of time, it does have an associated serious cost problem. If, for example, a half dozen intelligence compartments and two levels of security with several need-to-know restrictions are involved, the computer count could run to over a dozen. If, however, the data base had no need-to-know restrictions, was all of the same security level, and had only two compartments; only two computers would solve the problem. Multiple computers are not as serious a problem with the IAS because they represent a relatively small amount of the total system cost - small, relative to the data base and other peripherals. Still, this architecture is the most expensive.

1.6 Conclusions and Recommendations. The most important conclusion of the study is that it is feasible to design a secure data base system. This conclusion is based upon the findings of the data base guard approach and enciphered record and presupposes that the design of the data base guard, the multiported memory, would have a modest amount of software programs sufficiently small so that they could be certified. Such appears to be the case.

The conservative recommendation would be to go ahead with the design of the data base guard approach because there is a high probability that no penetration team would be able to gain unauthorized information. It is also likely that theoretical arguments could be developed to show that no unauthorized information could be obtained in any reasonable period of time and sufficient evidence provided to the Government to allow certification of the data base approach. While such a recommendation would assure the

demonstration feasibility, it would introduce a cost problem. The necessity for multiple computers and the associated core memory would preclude the retrofitting of existing systems.

An additonal reason for not recommending demonstration of the data base guard architecture is that little (or no) questions of feasibility exist. Multiported memories are well understood, have been built by Harris ESD and others, and are well within the state of the art. Building such a system would prove little or nothing.

Accordingly, the choice of alternative architectures to implement is narrowed to two.

- The clear record enciphered tag - This approach may or may not work. While it seems apparent that all but the most gifted criminals could be prevented from accessing unauthorized information, there is always the danger of some innovative tactic succeeding, such as bypassing exit guards, bribing or subverting maintenance personnel, and utilizing adminstrative traffic communication to convey information. This study reaches no conclusion as to whether the clear record/enciphered tag approach will guarantee that a user cannot access unauthorized information. Some progress has been made in assessing the details necessary to implement a dedicated, or retrofit, an IAS system. More work is needed.
  Because there is some doubt about the performance of this approach, it is not recommended for future development.
- Enciphered records - This second alternative has the appeal that - even though the computer environment may be viewed as hostile and users are exploiting flaws, trap doors, subversion of maintenance personnel, etc., - the computer criminal is denied access to the information although the enciphered data is freely accessible. From an engineering point of view, this approach is conceptually elegant. The problem of defining how, isoltion of RED-BLACK processors can be achieved has been accomplished. On this matter, the current study is positive. This leads to the recommendation that the design and implementation of the enciphered record approach also be pursued.

1.7 Compliance With the Statement of Work. The remaining portion of this report is organized into eight sections as described in the following paragraphs. The presentation is intended to conform to contractual requirements as set forth in the research and technology work statement.

Section 2, Background on the AN/GYQ-21(V), complies with Section 2 of the work statement. "The scope and proposed study is the IAS security protection in environments at HQ SAC, ADCOM and DIA" presents the description of the hardware and software of the IAS as well as its use in intelligence application.

Section 3, The Threat, complies with Paragraph 4.1.1.3 of the work statement. "Develop a strategy and tactics of malicious users. This task shall catalog ways of penetrating the IAS."

Section 4, Evaluation of Candidate Approaches, complies with Paragraph 4.1.1 of the work statement. "Examine various hardware/software approaches against actual security requirements and to recommend viable solutions using microprocessor technology. The security subsystem shall be an external control mechanism utilizing microprocesses."

Paragraph 4.2 of Section 4, Entrance Guards, complies with work statement Paragraph 4.1.1.4. "Examine the allowable sentence structures which may be composed by various users. The examination shall be utilized to build the query parameters allowable by operational intelligence analysts using the microprocessor."

Section 5, An Experimental, Enciphered, Data Base System, deals with two of the most promising appraoches, the enciphered tag and the encipered record. It is intended to comply with three sections of the work statement described as follows:

- Paragraph 4.1.1, "Design a Monitory and Control Unit that will prevent any accidental release of messages."
- Paragraph 4.1.1.2, "Design a Monitory and Control Unit that will prevent malicious users from damaging or compromising intelligence information in the AN/GYQ-21(V)."
- Paragraph 4.1.1.5, "The contractor shall design a malicious user protection mechanism and perform the synthesis over the detection algorithm and appropriate devices."

Section 6 of the report, RED-BLACK Processing, treats the problem of isolating the black processor from the red processor which has the capability of deciphering the records. It is a critical problem in the utilization of the enciphered record approach.

Section 7, Data Base Guard Approach, treats in some depth the problem of designing and developing a multiported memory for the multiprocessing configuration to achieve compartmentalized security.

Section 8, Findings, Conclusions and Recommendations, presents the results of the study.

Section 9, Specification and Security Monitoring subsystem complies with two sections of the research and technology work statement. Section 1, "The objective of the proposed program is to develop a security monitoring subsystem for the AN/GYQ-21(V) interactive analysis system (IAS). This initial effort is a study of evaluating the technical feasibility and development of the detailed specification of the prototype fabrication."

This chapter also complies with Paragraph 4.1.2, "Develop the Final Design for the Microprocessor. It shall consider the tasks of Paragraph 4.1.1 and based upon these tasks shall select an optimum approach based upon performance, cost and requirements."

SECTION 2.0

AN/GYQ-21(V) BACKGROUND

2.0 AN/GYQ-21(V) BACKGROUND

Scope of this study is limited to "Interactive Analysis Station (IAS) security protection in environments of Headquarters Strategic Air Command (HQ SAC), Aerospace Defense Command (AECOM), and the Defense Intelligence Agency (DIA)." This narrows the consideration of hardware, software and applications.

The central processing unit discussed in Paragraph 2.1 is limited to the Digital Equipment Corporation's PDP-11 series computers. Models 11/45 and 11/70 are in use in AN/GYQ-21(V) applications. The 11/60's are planned and undoubtedly others will be employed as they are announced. The PDP-11 product line is a dynamic one which is almost a decade old with continuing announcements of improved products.

The IAS also utilizes standard software. Up until now the RSX-11D, one of several operating systems developed and commercially marketed by the Digital Equipment Corporation is used in the IAS. Like all operating systems, RSX-11D is both large and vulnerable to penetration. It is discussed in Paragraph 2.2.

The applications of the PDP-11 to military problems are many and varied. It is extensively used in intelligence: it is a standard for the World-Wide Military Command and Control System; it is used for telecommunications, instrumentation, weapons control, scientific research, business data processing and many other applications. The scope of the study specifically limits the consideration of applications to those at the Defense Intelligence Agency, the Aerospace Defense Command and HQ SAC. These are described in Paragraph 2.3.

Paragraph 2.4 describes a typical configuration which incorporates the hardware/software and applications described in the preceding section. It is concluded that a data base system incorporates all the features of the intelligence applications, both existing and planned.

2.1 PDP-11/45. The source of information on the IAS hardware central processing unit is contained in the booklet entitled, "PDP-11/45 Processor Handbook" published by the Digital Equipment Corporation, Maynard, Massachusetts. Other models of the PDP-11 used in the IAS are essentially similar to the PDP-11/45. Chapter 6, Memory Management, describes the controls. In essence, there are three states in the IAS, each having different privileges. There are user states which may be occupied by several different users, each having different memory access privileges. There is a supervisory state, and, finally, the kernel state. The .ernel state is the most privileged and freely accessed and controls all information.

Table 1 describes the capabilities of the three states. The user state is limited in the area of memory it can address. The limitation is a specific number of virtual pages. Further, the user state is prohibited from exercising certain instructions, among them are input/output instructions, instructions assigning memory bounds, and instructions to change the state.

TABLE 1. IAS STATE CAPABILITIES

Mode	Ability To Address Memory	Instructions Prohibited	Trap, Abort Or Interrupt
User	Limited to assigned pages	Input/output Assign memory bounds Change state	
Supervisory	Limited to assigned pages	Assign memory bounds Change state	
Kernel	No limitation	No limitation	Transfer control to kernel state

Further note traps, aborts, or interrupts usually result in transition to the kernel state.

The supervisory state is also limited to assigned pages, but it can execute input/output instructions. It cannot change state or assign memory bounds. The kernel state has unlimited access to instructions and memory. It exercises control over the computing process by transferring control to the kernel state after traps, aborts, and interrupts.

The accesss controls to memory are specified by the access control field (ACF) of the Page Descriptor Register. Table 2 shows the access privileges.

The basic flaw in the hardware control is that if the malicious user can somehow enter the kernel state, all controls are bypassed and the user has free command of the system.

2.2 RSX-11D. The IAS software falls into two catagories: the RSX-11D Operating System, and the applications program. The Applications Program runs in the user mode and is not usually exploited by the computer criminal. By contrast, RSX-11D runs in the supervising or kernel mode and is usually the target of attack.

RSX-11D is the standard operating system used with the IAS. Like all operating systems, it exercises executive control over the computing facilities with the object of maximizing throughput via multiprogramming.

TABLE 2. CONTROL ACCESS PRIVILEGES		
Access Code	Mode	Function
000	Nonresident	Abort all accesses
001	Read-only	Abort on write attempt memory management trap on read
010	Read-only	Abort on write attempt
011	Unused	Abort all accesses - reserved for future use
100	Read/write	Memory management trap upon completion of a read or write
101	Read/write	Memory management trap upon completion of a write
110	Read/write	No system trap/abort action
111	Unused	Abort all accesses - reserved for future use

2.2.1 Functions. The principal functions of the IAS operating system are:

- Executive
- System generation
- Input/output operations
- Utilities
- Operating procedures
- Compliers

2.2.2 References. Details on the performance of these functions are contained in the following documents:

- Introduction to RSX-11D
- RSX-11D Concepts and Capabilities
- RSX-11D System Generation Reference Manual
- RSX-11D Input/Output Operations Reference Manual
- RSX-11D Utility Operations
- RSX-11D Operator Procedures
- RSX-11D Task Builder Reference Manual
- RSX-11D On-Line Debugging Techniques
- RSX-11D Device Handler Reference Manual
- RSX-11D Guide to Writing a Device Handling Task
- RSX-11D Macro Assembler Reference Manual
- RSX-11D FORTRAN IV Compiler
- RSX-11D System Test Reference Manual

2.2.3. The RSX-11D Executive. The executive of the operating system has five principal functions as follow:

- Memory management - The Executive is responsible for the allocation of core memory to programs and transfer of programs between core and disk file.
- Control of task execution - Multiprogramming is conducted in a series of tasks, several of which may be simultaneously resident in core memory. The tasks are prohibited from utilizing the input/output equipment or from communicating with each other. These services are performed by the Executive.
- System directories and systems lists - The Executive maintains parameters necessary for operation.
- Control of input/output operations - This critical function performed by the Executive includes controlling the reading and writing of disk files.
- Monitoring the console.

2.2.4 Input/Output Operations. Because the Executive commands the device handlers, and, accordingly, is able to print information out, store, and retrieve it from bulk memory sources, it has the power to bypass all access controls. It is perhaps more accurate to say that the control of input/output operations is, in fact, the access control mechanism. It will be demonstrated subsequently that one of the tactics utilized by the computer criminal is to enter the operating system and create commands for input/output.

2.3 IAS Application at HQ SAC, ADCOM and the DIA. The applications of the IAS are as the name implies controlling traffic to and from intelligence analysts and their terminals. The IAS is an essential link in the communication between man and machine. In general, the IAS is a component of a larger network.

2.3.1 Intelligence Networks. The National Military Intelligence Center is an important application of the IAS because 10 such machines are used and the process is accomplished entirely by the AN/GYQ-21(V). However, there are many other different types of intelligence installations. Figure 1 shows some of the intelligence installations of IAS as used by the Unified and Specified Commands and the Services. The dotted oblong in the center of the diagram encloses all of the computers resident at the Defense Intelligence Agency (DIA). The upper portion of the DIA blocks represent the National Military Intelligence Center.

Outside of the DIA blocks are the installations of the Unified and Specified Commands feeding information into the DIA. At the upper left-hand corner is the Alaskan Command (ALCOM). The lower left-hand corner is the Pacific Command (PACOM) and the center left is the Continental Air Defense Command (ADCOM) The Strategic Air Command (SAC) is shown to the left of the DIA box.

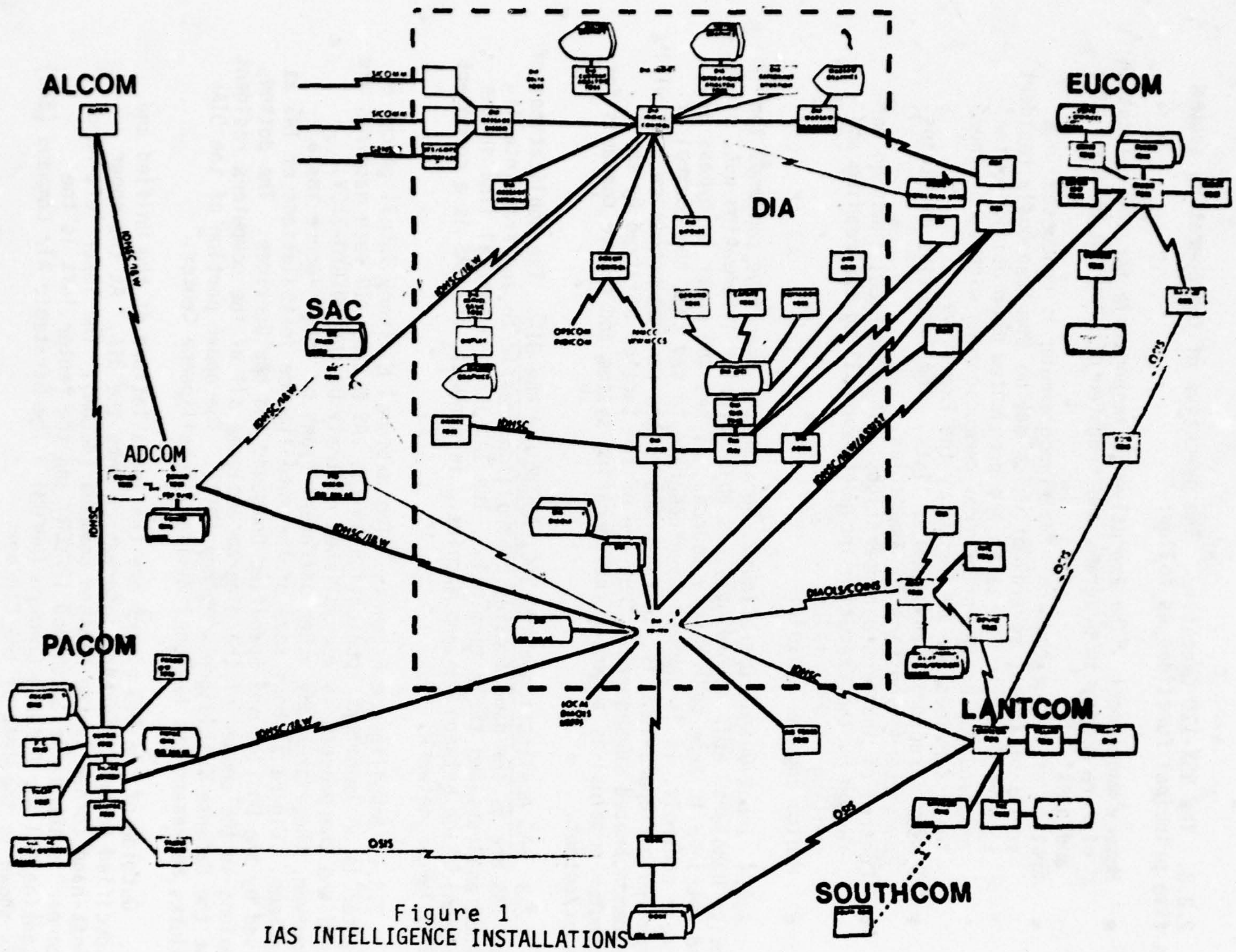


Figure 1
IAS INTELLIGENCE INSTALLATIONS

The upper right-hand corner indicates the equipment configuration at the European Command (EUCOM). And the Atlantic Command is shown on the lower right with a tentative link into the Southern Command.

The major links between the Unified and Specified Commands and the DIA are shown as IDHSC/IW, that is, there are communications links for the Intelligence Data Handling System Communications/Indications and Warning. In some instances the links already exist and are shared by the two applications. Other links are only contemplated.

The intelligence data handling system is comprised of several dozen large computers of various types. A commonly used computer is the Honeywell 600 or 6000 series. Frequenty, the IAS is used in conjunction with the larger machines. All IDHS sites contain a large data base associated with the general purpose computers.

The applications of the IAS are twofold: first, there is the stand-alone interactive data base system such as the National Military Intelligence Center (NMIC), shown in the upper center of Figure 1. Secondly, the IAS is used as a communications processor for a large Intelligence Data Handling System, (IDHS) computer.

2.3.2 Stand-Alone, Data Base Systems - NMIC. The focal point in the Pentagon for indications and warnings is the National Military Intelligence Center. NMIC is under development and utilizes 10 AN/GYQ-21(V) computers. The basic function of NMIC is the distribution and recall of incoming messages to intelligence analysts.

Figure 2 shows an overview block diagram of the NMIC support system processors with intercommunications links for normal and failure back up modes. The backup modes are shown as dotted lines. The processors are consecutively numbered 1 through 10 and are arranged in tandem to facilitate failure recovery.

Processors 1, 2, 3, and 4 form the message support subsystem with their supporting peripheral equipment. Principal among these are the data bases. There is a control subsystem shown as the No. 5 Processor which funnels all messages between the users in their origin and destination. The communications processor, INDICOM, is No. 6, and terminates the incoming circuits as well as providing external communications to other intelligence systems. The user support subsystem is comprised of two processors, No. 7 and No. 8, which support the analysts. Half of the analysts are on each of the two processors.

Figure 3 shows Processors No. 1 and 2 with their accompaniment of peripherals. The dissemination analyst bus carries data from Processor No. 1 and interconnects to the following peripherals and memory:

- 120K core memory
- 132 column 1000 line per minute line printer
- LA36-CA teletype

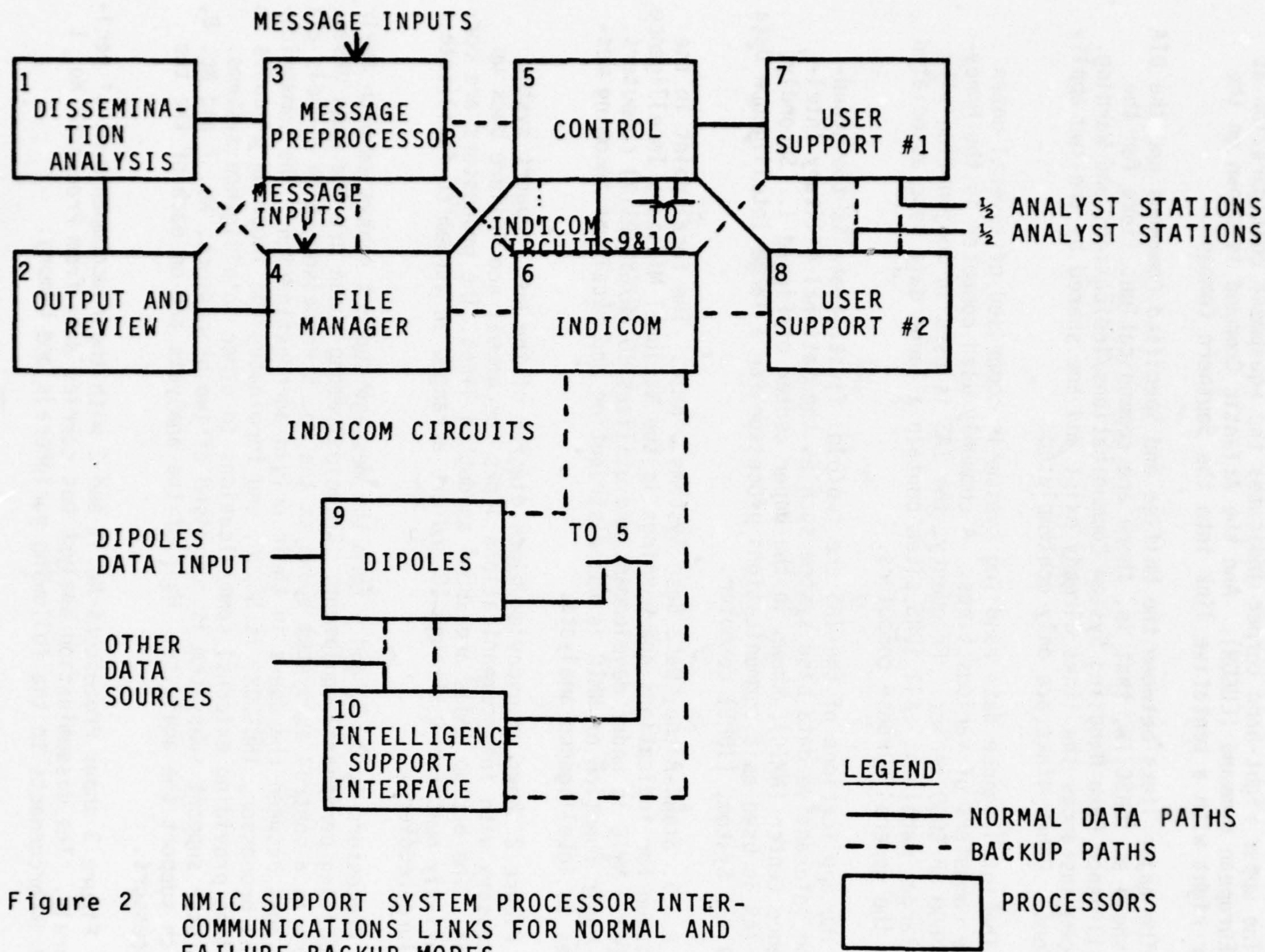


Figure 2 NMIC SUPPORT SYSTEM PROCESSOR INTERCOMMUNICATIONS LINKS FOR NORMAL AND FAILURE BACKUP MODES

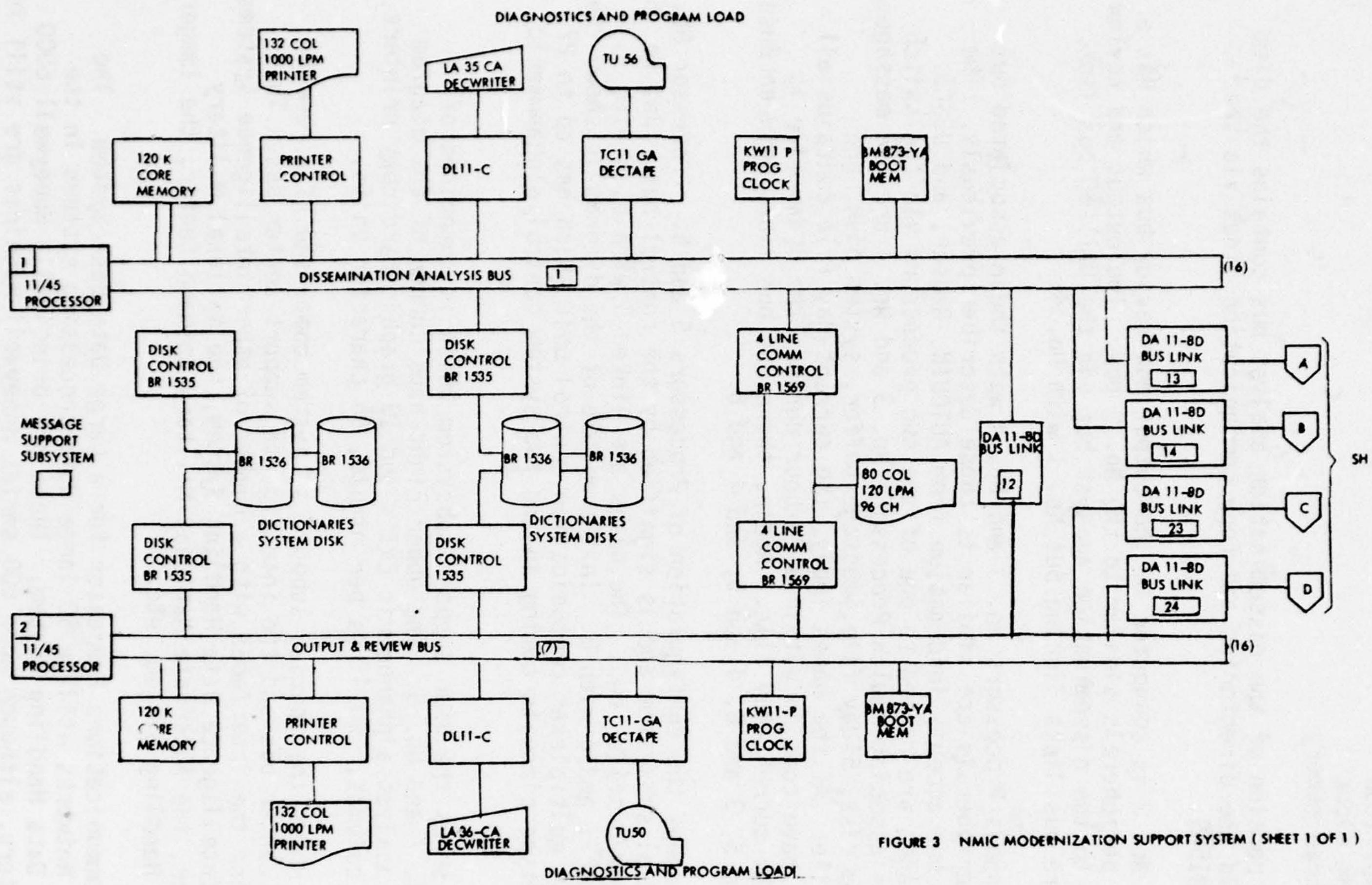


FIGURE 3 NMIC MODERNIZATION SUPPORT SYSTEM (SHEET 1 OF 1)

- DEC tape TU56
- Program clock
- Bootstrap memory

The lower portion of the dissemination analyst bus contains the disk files that hold the directories and four communication lines via the multiplexer BR1569.

Processor No. 2 is connected to the output and review bus which has a complement of peripherals similar to the No. 1 bus. The output and review bus is linked to the dissemination analyst bus via the DA11-BD bus link. Also, there are bus links linking Bus No. 1 with No. 4.

Figure 4 shows Processors No. 3 and No. 4 with their associated bus links. The peripherals are similar to those described previously. The message input bus accepts information from AUTODIN, Genser, and DSSCS. Incoming messages are routed to one of the two processors via the switch. The data bases associated with Processors No. 3 and No. 4 are the message buffer, system disk, 5-day file, message buffer, system disk, and current-day file. As the names imply, the current-day file contains all messages that have come in within a 24-hour period; the 5-day file is archival to the current-day file. Again, there are bus links between Buses 3 and 4, 3 and 5, 3 and 6, 4 and 5, and 4 and 6.

Figure 5 shows the configuration of Processors 5 and 6. Processor No. 5 is the control subsystem and is signified by the control bus. Bus No. 6 is the intercom/profiler bus. The buses are linked 5 with 6, 5 with 7, 5 with 8, 6 with 7, and 6 with 8. In the middle of the diagram is shown the 32 line BR1569 multiplexer communication control unit which has up to 27 INDICOM teletype circuits coming in and two system control alphanumeric CRT's.

Figure 6 shows the user support subsystem which is comprised of Processors No. 7 and No. 8. The upper right-hand corner of the diagram shows the 31 analyst alphanumeric CRT's and 10 graphic hard copy printers, these are 80 columns, 120 lines per minute, 96 character units.

Figure 7 shows the special support subsystem comprised of a remote intelligence center bus and the intelligence support center bus. This system contains the interfaces with a number of other intelligence systems, such as the Intelligence Data Handling System, the National Military Command Center, the Alternate National Military Command Center, the Imagery Related Data Handling System, etc.

2.3.3 Communications Processor for a Large Data Base System. The Intelligence Networks utilize 50 large data processing systems in the Intelligence Data Handling System. These are principally Honeywell 6000 series computers, although some 600 series Honeywell machines are still in use. The typical application of the IAS is to act as a communications processor between these large machines and the user terminals.

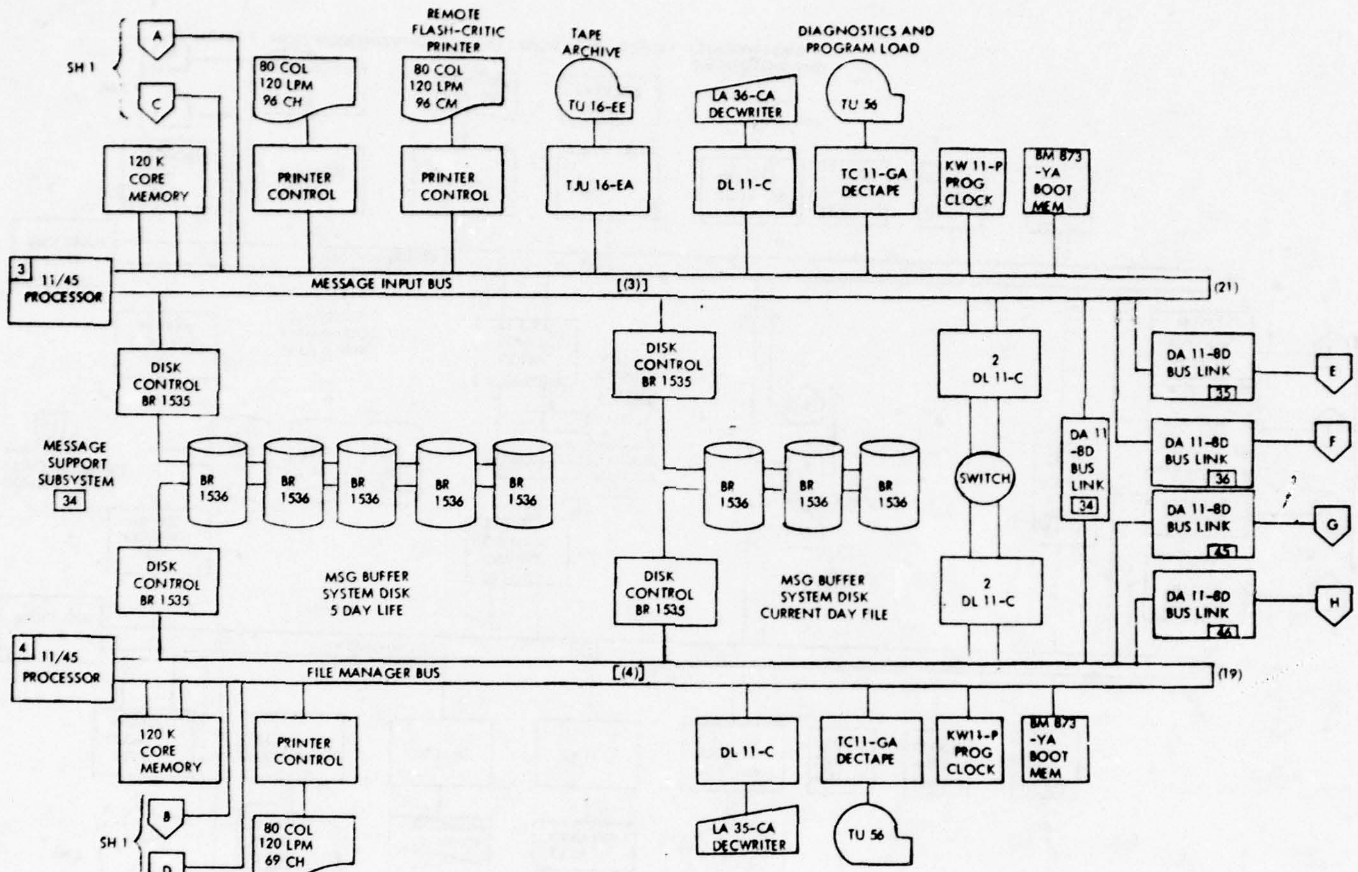


Figure 4 NMIC MODERNIZATION SUPPORT SYSTEM (SHEET 2 of 5)

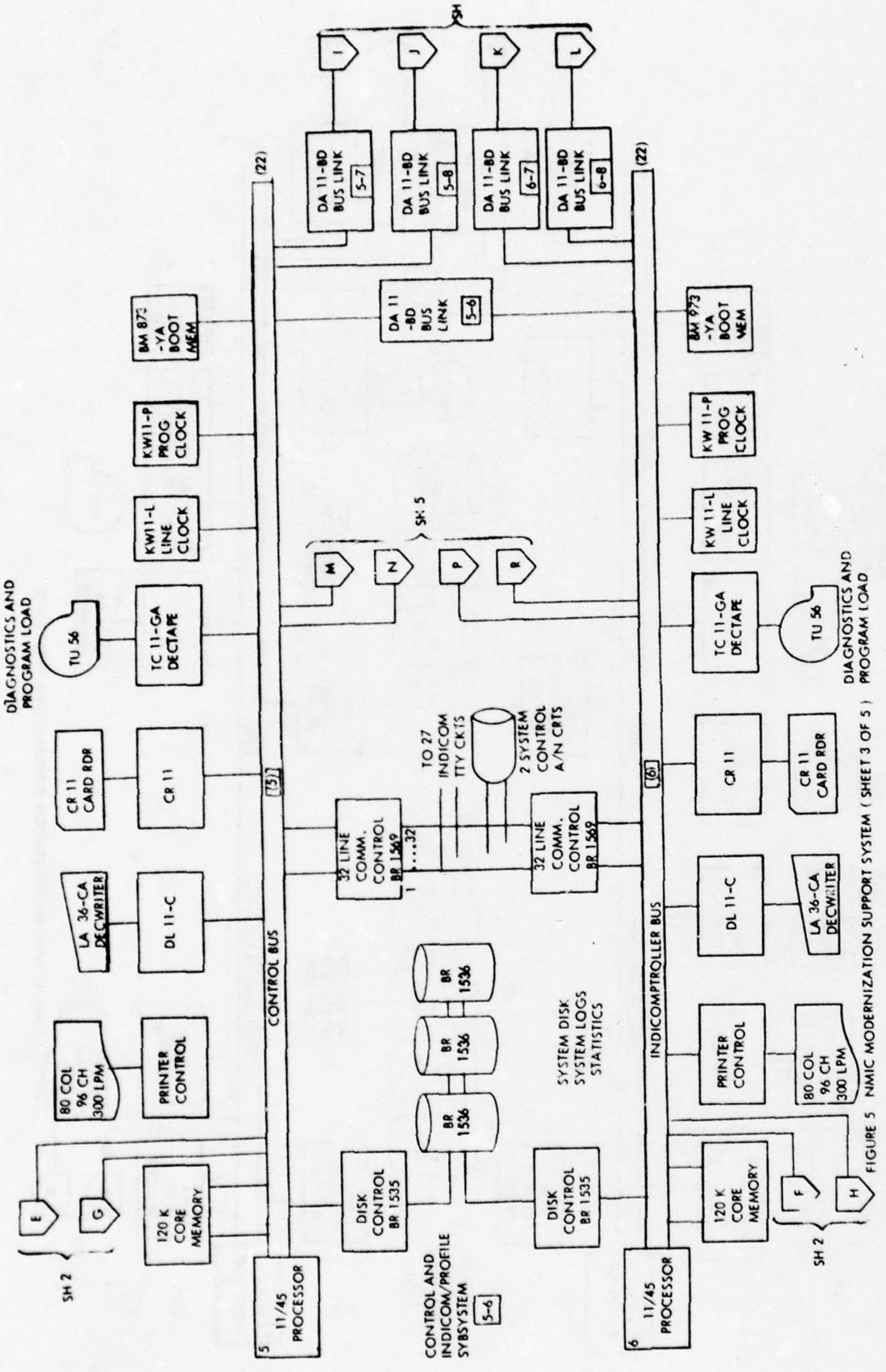


FIGURE 5 NMIC MODERNIZATION SUPPORT SYSTEM (SHEET 3 OF 5)

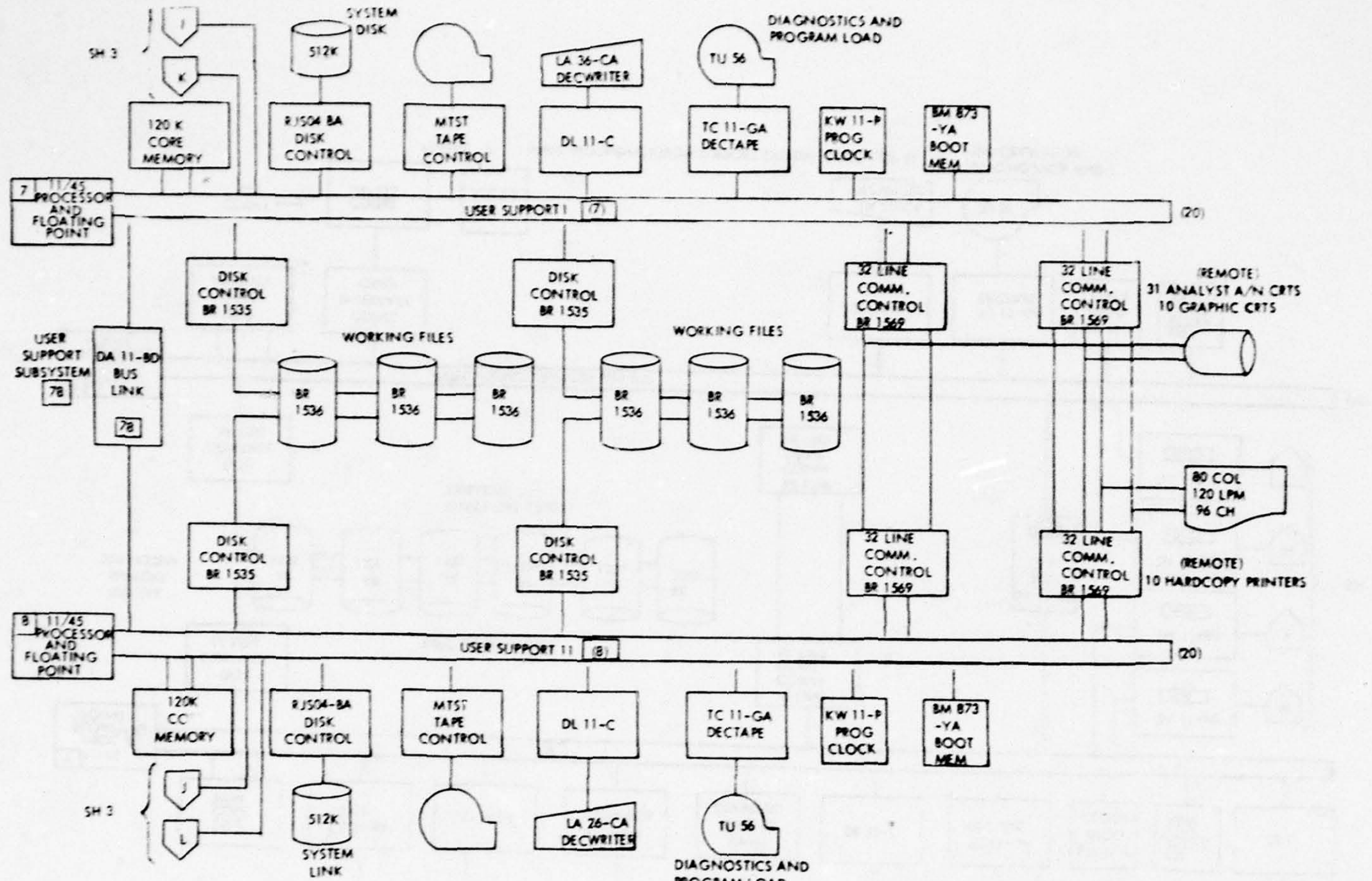


FIGURE 6 NMIC MODERNIZATION SUPPORT SYSTEM (SHEET 4 OF 5)

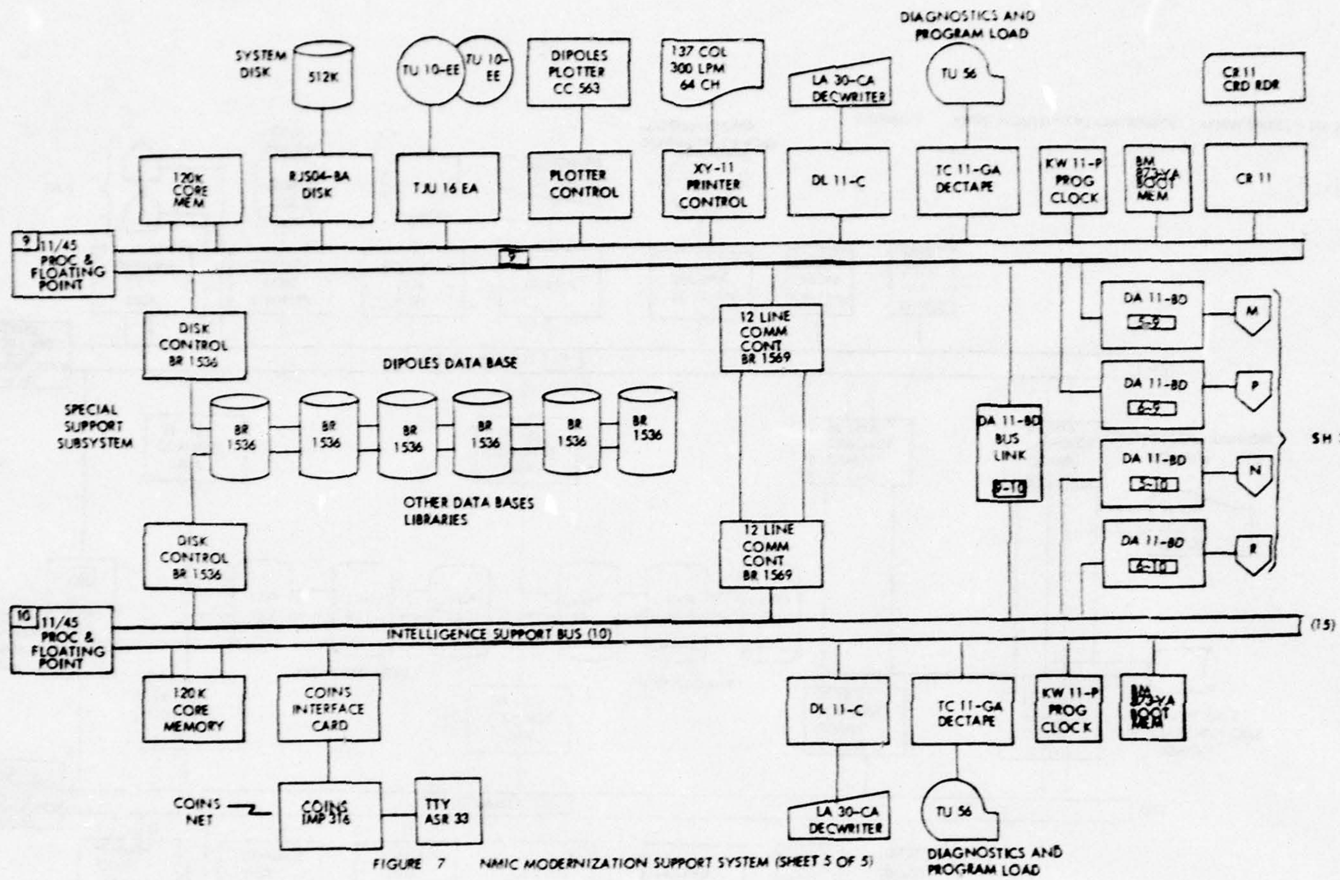


FIGURE 7 NMIC MODERNIZATION SUPPORT SYSTEM (SHEET 5 OF 5)

2.4 Typical Configuration of the IAS. There is no such thing as a "typical" AN/GYQ-21(V) configuration. NMIC is different from DIAOLS which is different from PACER, etc. There are perhaps no two applications that are identical. However, from a functional point of view, one configuration incorporates the essential features of the diverse applications. Figure 8 illustrates such a configuration. In general, there are more than one computer and data base involved. In the figure, two are shown. Each of the two computers has its own data base and set of remote terminals. A terminal user of Computer No. 2 can interrogate the files, that is, Data Base No. 2, as well as Data Base No. 1, via a telecommunications link, linking the two computers. In a similar way, a terminal user of Computer No. 1 can interrogate the files of both Computers 1 and 2.

The user control problem can now be stated. The user of a terminal is denied access to any records in either data base that without specific authorization, that is, security level, need-to-know, and intelligence compartment. The user is allowed free access to all other records.

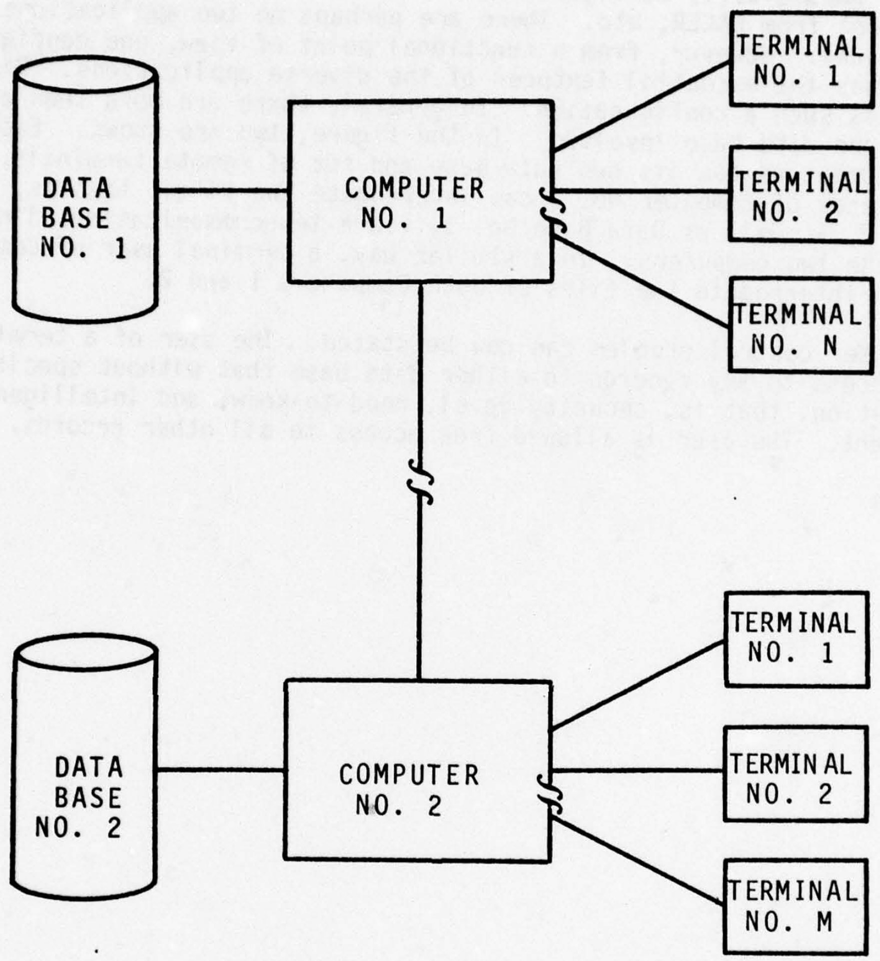


Figure 8 IAS ESSENTIAL FEATURES

SECTION 3.0

THREAT

3.0 THREAT

The primary measure of performance of a secure data base system is the ability to withstand a determined attack by a well-financed, highly-skilled enemy for a period of years, if not decades. To build such an impenetrable data base system requires a detailed knowledge of the tactics employed by a malicious user. What are the tactics that should be tried against candidate architectures? Fortunately, there are a number of books, papers and other communications on the subject. The "Computer Security Institute" periodically publishes reviews of books and papers on computer crime. A reader interested in a literature search on the subject is referred to Dr. Saltzer's article: "On-Going Research and Development on Information Protection."[3] In this article on Page 23, Dr. Saltzer gives an index of abstracts on the subject and on Page 9, he identifies organizations having a capability in system penetration exercises. Among those listed in the article are Lawrence Livermore Laboratory, Information Sciences Institute, IBM Corporation Research Division, AF Electronics Systems Division and the Systems Development Corporation.

Unfortunately, the reports of criminal activity and the precise methods of the penetration teams tend not to get published. There is an understandable reluctance to publish criminal methods. There is one exception to this general rule and it is contained in a document that sets forth specifics on penetration tactics successfully used against MULTICS.[4] That presentation is abstracted in part in this chapter.

Criminal attacks upon computer systems are pursued in two stages. First, the criminal gains command of the system by exploiting hardware flaws, software flaws or trapdoor. Next, the criminal proceeds to seize the protected data by use of any one of several tactics.

3.1 <u>Criminal Tactics Given Command of the System</u>. Suppose a User at a NMIC terminal somehow acquired the ability to read or write any word in PDP-11 memory. Subsequently, in this section it will be shown that it is quite easy to acquire such a capability. The criminal could then proceed with any or all of the following tactics:

- Dump the password file. Intelligence systems vary in the exact method used to log a user on the system initially. However, the procedure is generally as follows. The User requests the use of the system. The User is identified by name, social security number, and other identifiers. The system then requests a password which is a series of alphanumeric characters. If the

[3] Saltzer, Dr. Jerome H., <u>Operating Systems Review</u>, Vol. 8, No. 3, July 1974.

[4] Krager, P. A. and Schell, R. R. "Multics Security Evaluation: Vulnerability Analysis," AF Electronics Systems Div., ESD-TR-74-193, Vol. 2.

User provides the proper sequence, the computer assumes that the User is satisfactorily identified and is granted access. In the computer system is a file of passwords associated with each legitimate user. The malicious user can locate the start of the file in using the acquired ability to read data and read out the password file. With this capability the criminal can then masquerade as a user with higher privilege.

This tactic is frequently, though not always, countered by enciphering the password file. Still, if the code can be broken, the user can gain access to any information in the system.

- Falsify the access control list. Located beside the individual's identification is a list of access privileges. For example, one user might be permitted access to all Secret files while a user with higher privilege would be allowed access to all Secret plus all Top Secret files. The malicious user, by gaining access to the access control list, could change a privilege from Secret to Top Secret.
- Masquerading as a user with higher privilege. There is usually a word contained somewhere in the PDP-11 memory that identifies the current user of the system. Access privilege is checked against this individual. Using the capability to alter information anywhere in storage, the malicious user can change an identification to a user with higher access privilege.
- Falsifying logs. In the National Military Intelligence Center and other Intelligence Systems, all transactions are logged. The malicious user utilizing the capability to read and write memory, can falsify any logs and erase incriminating evidence.

There would be some difficulty in finding the critical words to read or write, but in practice this difficulty has not proved insurmountable. Listings are generally unclassified and are available. Even if this were not the case, locating critical words by trial and error can be accomplished in hours or days. Certainly not years or decades as prescribed by the performance criterion.

In addition to reading or writing, if the terminal user has somehow gained the ability to execute a set of instructions, beginning at some prescribed address, the criminal can also successfully execute the following tactics:

- Command the operating system to dump priviledged data. The object of the attack is generally records stored on disk file. The disk file, in turn, is controlled by a device handler running under the operating system RSX-11D. By placing key w rds in the device handler and transferring control, the malicious user can proceed to dump any and all information in the data base.
- Write and execute programs that dump information in the data base. Even if the malicious user cannot locate the device handler, programs can still be written and executed to perform this task, thereby gaining access to all information contained in the data base.

The preceding list is by no means complete, but it will serve to illustrate tactics that can be effective for an espionage agent.

The criminal can also crash the system by writing into the operating system and by other means. Such sabotage attack is not in our view, considered serious, because service would be only denied a brief period of time. For example, in the National Military Intelligence Center, copies of the operating system are available on line and can be reloaded in a matter of minutes. Attempting to crash the system several times would place the criminal in a dangerous situation. For this reason, the study does not consider the problem of preventing a sabotage attack. Indeed, this is thought to be improbable.

The conclusion is obvious. If there is any way that the criminal can gain the power to read and write any one word of main memory, the time required for penetration of unauthorized data will be minutes or even seconds; rather than decades as prescribed by performance.

3.2 Gaining Command by Exploiting a Trapdoor. Trapdoors are unnoticed pieces of code which may be inserted into a system by a penetrator. The trapdoor would remain dormant within the software until triggered by the agent. Trapdoors inserted into the code implementing the reference monitor could bypass any and all security restrictions in the system. Trapdoors can potentially be inserted at any time during software development and use.

3.2.1 Example of a Trapdoor in MULTICS. Check$Device-Name is a program that runs in Ring 0, the most priviledged ring in MULTICS. Figure 9 shows an abbreviated listing of the code in PL-1 and the trapdoor.

The program is called with three parameters. The key which is the 72-bit binary string, Pointer 1 which points to a 36-bit word, and Pointer 2 which points to another 36-bit word. The trapdoor is inserted at the end of the program where it would normally return before exiting.

> If key = bit____string____constant____password
> is the coding that compares the 72-bit binary
> word passed initially into the program with a
> 72-bit password.

If the two match, then the trapdoor program is executed. It appears on the following line.

then ptr2->overlay=ptr1->overlay.

The line of code simply says that the word to that ptr 2 points to is made equal to the word that ptr 1 points to. By swapping data between words pointed to, the trapdoor is able to read and write any word in memory, thereby allowing the malicious user to gain command of the system. Readers interested in details of the trapdoor are referred to Appendix C, Page 115 of the document referenced in Footnote 4 (Paragraph 3.).

```
check$device_name: procedure (a, b, code);

    declare 1 code parameter,
                  2 err_code fixed binary (35),
                  2 key bit (72) aligned,
                  2 ptr1 pointer aligned,
                  2 ptr2 pointer aligned;

    declare overlay fixed binary (35) based;

    /*  Start of regular code  */
                 ...;

    /*  Here check$device_name would normally return  */

              If key = bit_string_constant_password
                  then ptr2 ->overlay = ptr1 ->overlay;

              return;

         end check$device_name;
```

Figure 9. Trapdoor in Check$Device_Name

3.2.2 Data Base Query Trapdoor. The preceding example illustrated the basic principles of a trapdoor. There is a code which the user must know to trigger the door, but possessing this key allows the malicious user to gain command of the system. A limitation in the preceding example was that the word to be read out of memory is only transferred to another register and the word to be written is transferred from one memory location to another. In the NMIC, for example, as well as most other intelligence applications, the user does not have programming privileges, and, therefore, such a trapdoor would be of little use. However, this deficiency can be overcome quite easily by the data base query trapdoor.

Suppose there existed a trapdoor in NMIC that was triggered by some unlikely sequence of code typed by the user at a terminal, say QZKX. Suppose further that following the code the trapdoor would interpret the following instructions:

READ, (ADDRESS), causes the word to be typed on the terminal

WRITE, (ADDRESS) (NUMBER)

GO TO, (ADDRESS)

END.

With these commands the terminal user, even though being denied programming privileges, could fully command the PDP-11.

The trapdoor can be placed in the command interpreter software. However, in the case of NMIC, these are application programs, and the ability to read and write would be restricted to those prescribed by the application program. A powerful alternative would be to execute the trapdoor in the kernel state, and allow reading and writing in any part of the PDP-11.

3.2.3 Ease of Inserting and the Difficulty of Finding Trapdoors. Since the trapdoors are very small, sometines only one or two instructions, they are difficult or impossible to detect in the large software systems used in the IAS. Frequently, such software runs in the tens of thousands of instructions and at times, it is in excess of a hundred thousand instructions. The trapdoor may be hidden in the object code. When this happens, the listings give no indication that the trapdoor exists. This tactic can be countered by recompiling at frequent intervals from source listings. Even so, the trapdoor may be hidden in the compiler and automatically reissued when the programs are assembled.

The threat of trapdoor insertion is further complicated by the ease with which a foreign agent would have access to the programs. RSX-11D is developed in an uncleared facility in Maynard, Mass. Although the application programs are developed by cleared personnel, the listings themselves are unclassified. The programs are communicated by open mail and sometimes by telecommunications. These provide further opportunities for the foreign agent to insert trapdoors. Finally, there is the possibility of a Trojan horse, i.e., a program being offered which is seemingly desirable, yet contains a trapdoor.

The threat of trapdoor insertion is most serious. If it is placed in the application programs, it will do little harm because the programs run in the user state and have restricted access to memory. However, if the trapdoor is inserted in the supervisory state, the malicious user would be able to command the device handler for the disk file, and be able to read and write any information contained in the data base. Finally, the trapdoor may be inserted in the kernel state giving the user complete command over the system.

The only known way of certifying that trapdoors do not exist is to prove the software does only what it is supposed to do. This task has proven impossible. In the case at hand, RXS-11D occupies some 26,000 lines of code. It is impossible to certify such a complex program.

3.3 <u>Gaining Command of the System by Exploiting Hardware Flaws</u>. It will now be shown, by example, that it is relatively simple for the computer criminal to gain command of the system by exploiting hardware flaws.

The experiment abstracted here is described in detail in the document referenced in Footnote 3, Pages 17 through 22. The MULTICS system on which the experiment was run has a virtual memory divided into segments with each segment broken down into pages and words. Access to memory for a given user is controlled by allowing access to certain segments and disallowing access to other segments. The descriptor segment contains the access privileges of that user. Such privileges include READ ONLY, READ AND EXECUTE, and NULL ACCESS.

In the reported experiment, a program was written to test the effectiveness of the access controls. The program was run under operating conditions in the background, once each minute for a period of 1100 hours and tested illegal instructions and access to restricted memory space.

As a result of the experiment, two errors were found in the hardware. The first was an undocumented instruction that did not appear to compromise security. The second was far more serious. It permitted the execution of instructions to bypass the access controls under certain prescribed conditions.

The execution of the "execute instruction access check bypass" proceeds as follows. There are three segments involved: the X segment in which the current user has READ and EXECUTE privileges, the Y segment in which the current user has READ ONLY privilege, and the Z segment in which the current user is disallowed access.

The progam sequence is transferred to execute in Segment X. At a given location in this segment, an instruction is encountered that directs the execution of an instruction stored in Y 0. At Y 0 the instruction reads, "store the contents of the A register at location X 6." When location X 6 is read, there is an ITS command which directs the CPU to store the contents of the A register at location Z 0. Even though Z 0 is restricted to the user, the command is executed.

This vulnerability permits the user to read and write any word in a restricted memory location. As a result, the "execute instruction access check bypass" allows the user to initiate the tactics described in Section 3.1.

3.4 Exploiting Software Flaws. The reference in Footnote 4 sets as an objective finding one flaw each in hardware, software, and procedures. The software flaws were so plentiful that three were documented, Insufficient Argument Validation, Master Mode Transfer, and Unlocked Stack Base. These flaws are briefly described in the following paragraphs. Readers interested in detailed information can reference Footnote 4, Section 3.3, Software Vulnerabilities, Page 22. These examples are chosen to illustrate how vulnerable a large hardware/software system is.

3.4.1 Insufficient Argument Validation. Suppose there existed a subroutine that executed in a privileged mode, in the example given, the mode was Ring 0 of MULTICS. For simplicity here, we assume it runs in the

kernel mode of the IAS. Since the subroutine runs in the kernel mode, it has access privileges to any part of memory.

The calling program runs in a much less privileged mode, Ring 4 in the MULTICS example. For purposes of illustration here, consider operating in the user mode in the IAS. Further suppose the subroutine can be made to store a word supplied by the calling routine at an address also supplied by the calling program.

Clearly, such a situation is dangerous because the calling program can make the subroutine write in any location of memeory. For example, the calling program might choose to forge the user's identity by writing over the identity held in protected space. To prevent this from happening, on MULTICS every argument supplied by the calling routine is checked to see if the calling program has WRITE access privileges to the object address. This is termed Argument Validation.

Before discussing the vulnerability, it will be necessary to introduce two indirect modifiers used in MULTICS. The ITS modifier indirects the program by commanding DON'T STORE HERE, but rather in the location specified in the address field of this word. The IDC modifier (Increment Address, Decrement Tally, and Continue) redirects the location to that specified in the address field of the tally word after it has been incremented. Figure 10 taken from the document referenced in Footnote 4 illustrates the vulnerability. The argument supplied by the calling routine contains an IDC indirect modifier. The mistake in validation is to take the tally word address field before it is incremented rather than after. This leads to a first reference (shown at the right-hand side of the figure) which is checked by the validation routine. This seems to pose no severe problem because the address is just incremented by one from the second reference, which is the correct reference. However, if the two references are indirected a second time by an ITS modifier the object address can be in completely different parts of memory. Shown in the figure the second reference which is the correct reference directs to an argument which is writable only in Ring 0, while the first reference, the one checked by the validation routine, goes to an argument writable in the user ring.

Because of the insufficient argument validation, a user supposedly restricted to limited memory access can, in fact, write anywhere at all. Variation of this attempt would also permit the user to read any location in memory. With this capability, the malicious user could gain full command of the system.

3.4.2 Master Mode Transfer. In the MULTICS system, or rather the old MULTICS with ring-crossing software as implemented in the Honeywell 645, there existed a master mode procedure operating in Ring 4 called "Signaller." The function of Signaller was to communicate to the user certain types of faults. For example, if the user attempted to divide by zero, signaller would handle the fault by communicating the difficulty to the user. The result is that the Signaller program could be effectively

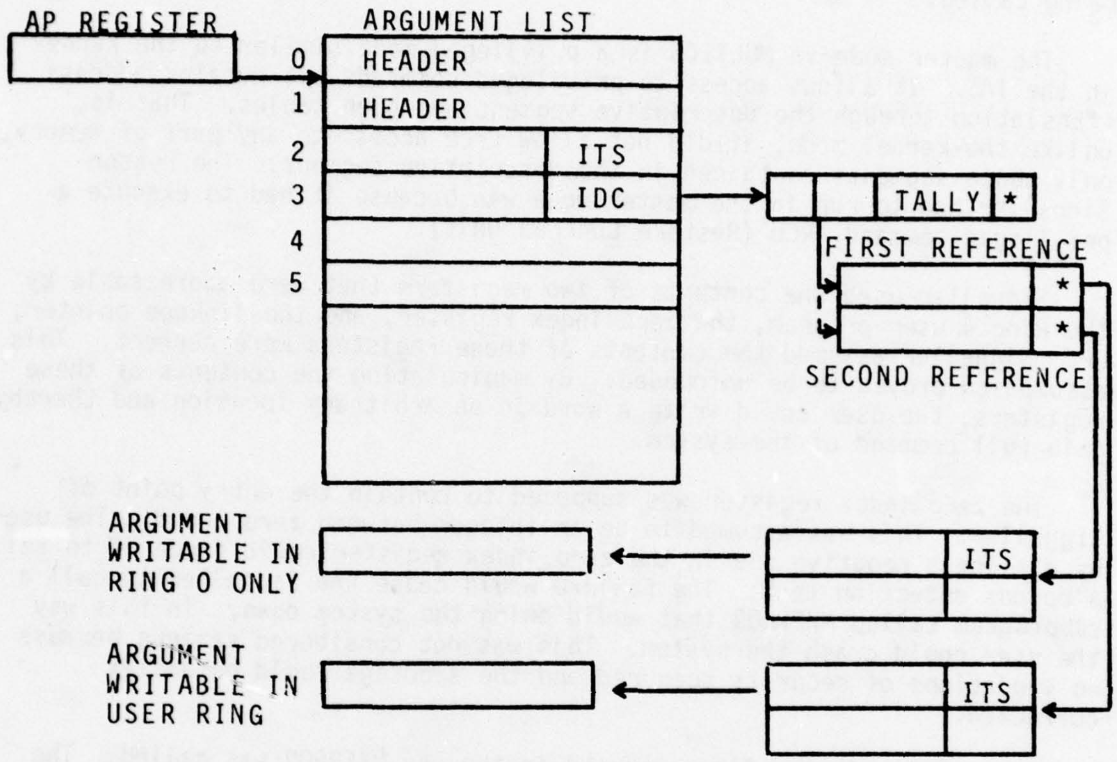


Figure 10 INSUFFICIENT ARGUMENT VALIDATION

called by the user program, also operating in Ring 4, by attempting division by zero or some other operation that would result in Signaller being called.

The master mode in MULTICS is a privileged mode similar to the kernel in the IAS. It allows access to privileged commands but retains address translation through the descriptive segment and page tables. That is, unlike the kernel mode, it did not allow free access to any part of memory, only those segments contained in the descriptive segment. The reason Signaller had to run in the master mode was because it had to execute a privileged command, RCU (Restore Control Unit).

Signaller used the contents of two registers that were addressable by the Ring 4 user program, the zero index register, and the linkage pointer, LP. Signaller assumed the contents of these registers were correct. This assumption proved to be unfounded. By manipulating the contents of these registers, the user could write a word in an arbitrary location and thereby gain full command of the system.

The zero index register was supposed to contain the entry point of Signaller. This was assumed to be an integer between zero and N. The user by placing a negative one in the zero index register could force it to fail a bounds detection test. The failure would cause the Signaller to call a subprogram called MXERROR that would bring the system down. In this way the user could crash the system. This was not considered serious because no violations of security occurred and the sabotage could be easily corrected.

The fatal software flaw occurred in the way MXERROR was called. The instruction was as follows:

```
tra lp|12,
```

that is, transfer to the address contained in the link pointer plus 12. Since the contents of the link pointer could be written by the user program, the user force Signaller to write a word in an arbitrary location still in the master mode. For example, the user can write the contents of the A register into a segment descriptor word that, in effect, would give the user program the ability to write into an arbitrary segment. In this way, a user can gain full command of the system.

3.4.3 Unlocked Stack Base. As seen in the preceding example, an important method of exploiting software flaws comes from the fact that the system registers, such as the link pointer (lp) are addressable by the user, yet privilege procedures assume, incorrectly, they point to prescribed locations. The unlocked stack base exploits a second register, the stack pointer. There are 8 pairs of registers in MULTICS each containing 18-bit pairs. Lp is the link pointer; lb is the linkage base; sp is the stack pointer, and sb the stack base. Originally, the stack base was locked, that is, it could not be changed except in master mode. Because

this proved to be inefficient, the stack base was unlocked, resulting in the ability of a user-mode program to change it arbitrary.

Figure 11 shows the listings for Signaller. It will be recalled that Signaller checks the contents of index register zero. When it finds a minus one in the register, it concludes the integer is out-of-bounds, saves the registers, and calls MXERROR, shown in the listing as tra lp|12, since lp|12 is assumed to point to MXERROR which will bring the system down. The second flaw occurs in the coding that stores the register. In particular, sreg sp|10 stores the index registers beginning at the location specified in sp|10. As a result of this, one register, specifically the AQ register, is transferred to the address contained in sp|14. If a user changes the sp register, the net result stores the contents of the AQ register at any location specified.

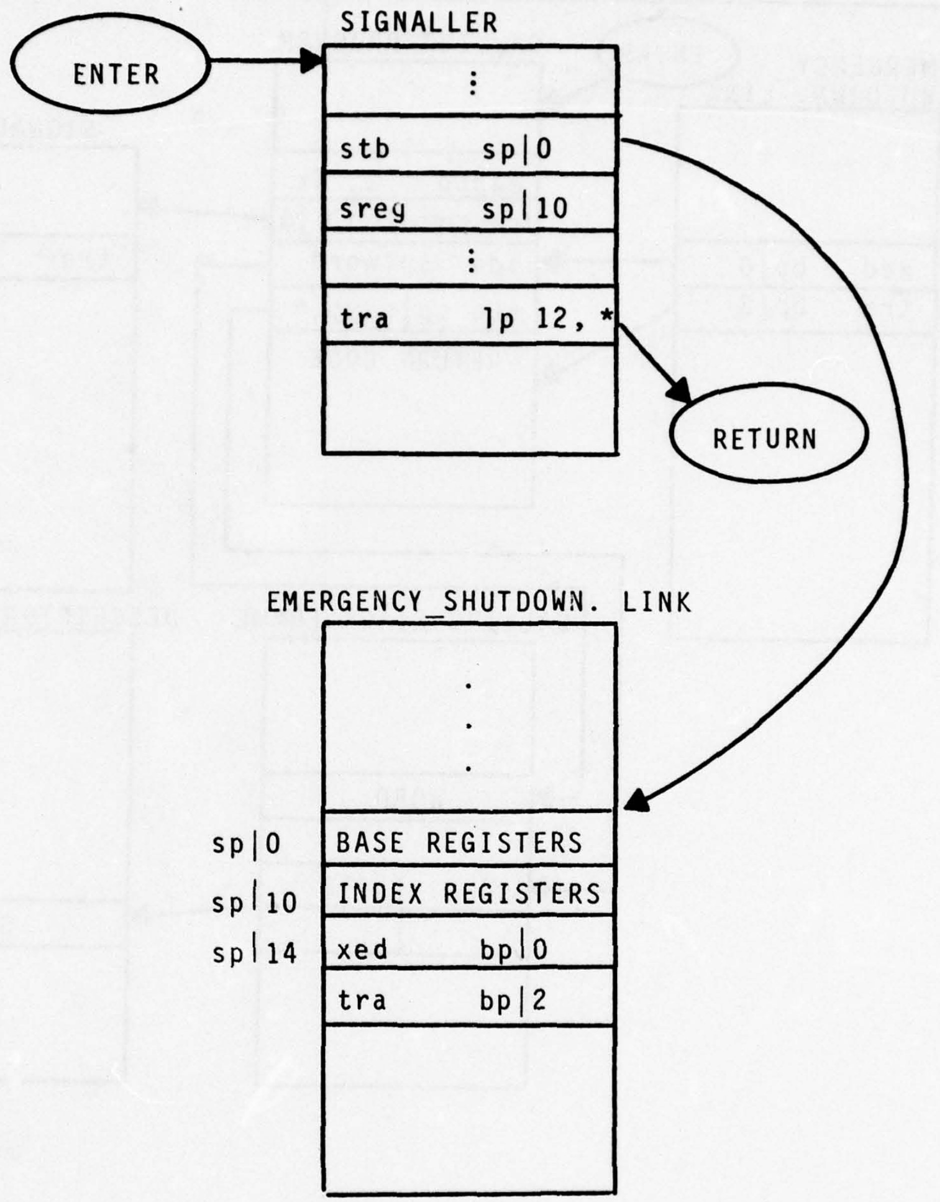
Figure 12 illustrates the operation of storing in an arbitrary location. The execution of the Signaller program results in storing the contents of the AQ register at location sp|14. The contents of the AQ register are set at exd bp|0; tra bp|2, that is, the command is execute double; the instruction whose address is contained in bp|0. Then tra to the instruction whose address is contained in bp|2. In the figure, it is shown that the trapdoor contained in sb|14 is placed in the emergency shutdown link. This is a segment which is seldom used by the system and it is convenient point to store the trapdoor.

Figure 13 shows how the trapdoor is used. The calling program is entered and sets up the base pointer register. Next, the calling program transfers control to Signaller. The lp register has been set so that when Signaller tries to call MXERROR, assumed to be at lp|12, it in fact calls the emergency shutdown link location where the trapdoor is held. The execute double instruction of the trapdoor directs to an instruction that commands the Q register to be loaded with the word from the calling stack frame. The second of the double instruction directs that the word stored in Q be stored at a prescribed location which happens to be in the calling frame. The prescribed location contains an indirect modifier which directs in the example the word be stored in the descriptive segment thereby forming a new segment descriptor word. The use of the trapdoor allows the calling program to write any arbitrary word anywhere in the system. The point at which the word is chosen to be written is a segment descriptor word which now gives the user privilege to access the prescribed segment. Again, this may be a segment that contains a user identification; therefore, giving the current user the right to modify his identity and again gain command of the system. The trapdoor allows the user to read and write an arbitrary word in the system, thereby gaining full command of the system.

3.5 <u>Penetration of RSX-11D</u>. The major portion of this chapter is drawn from experiences reported by penetration teams. However, Harris did conduct its own penetration tests on RSX-11D during the course of the study. The test utilizing Method 1, described below, succeeded in

```
          cmpx0     2,du              "call in bounds?
          tnc       transfer_vector,0   "Yes, go to entry
          stb       sp|0              "Illegal call here
          sreg      sp|10             "save registers
          eapap     arglist           "set up call
          stcd    sp|24
          tra     lp|12,*             "lp|12 points to mxerror
a:        code
          ...
b:        code
          ...
transfer_vector:
          tra       a
          tra       b
          end
```

Figure 11. Master Mode Interpreted Object Code (Signaller)



SETUP CONDITIONS

```
AQ REGISTER   :=xed bp|0; tra|bp 2
INDEX 0       :=-1
sp            :=ADDRESS (UNUSED STORAGE IN EMERGENCY_SHUTDOWN.LINK)
lp|12         :=ADDRESS (RETURN LOCATION)
```

Figure 12   UNLOCKED STACK BASE (STEP 1)

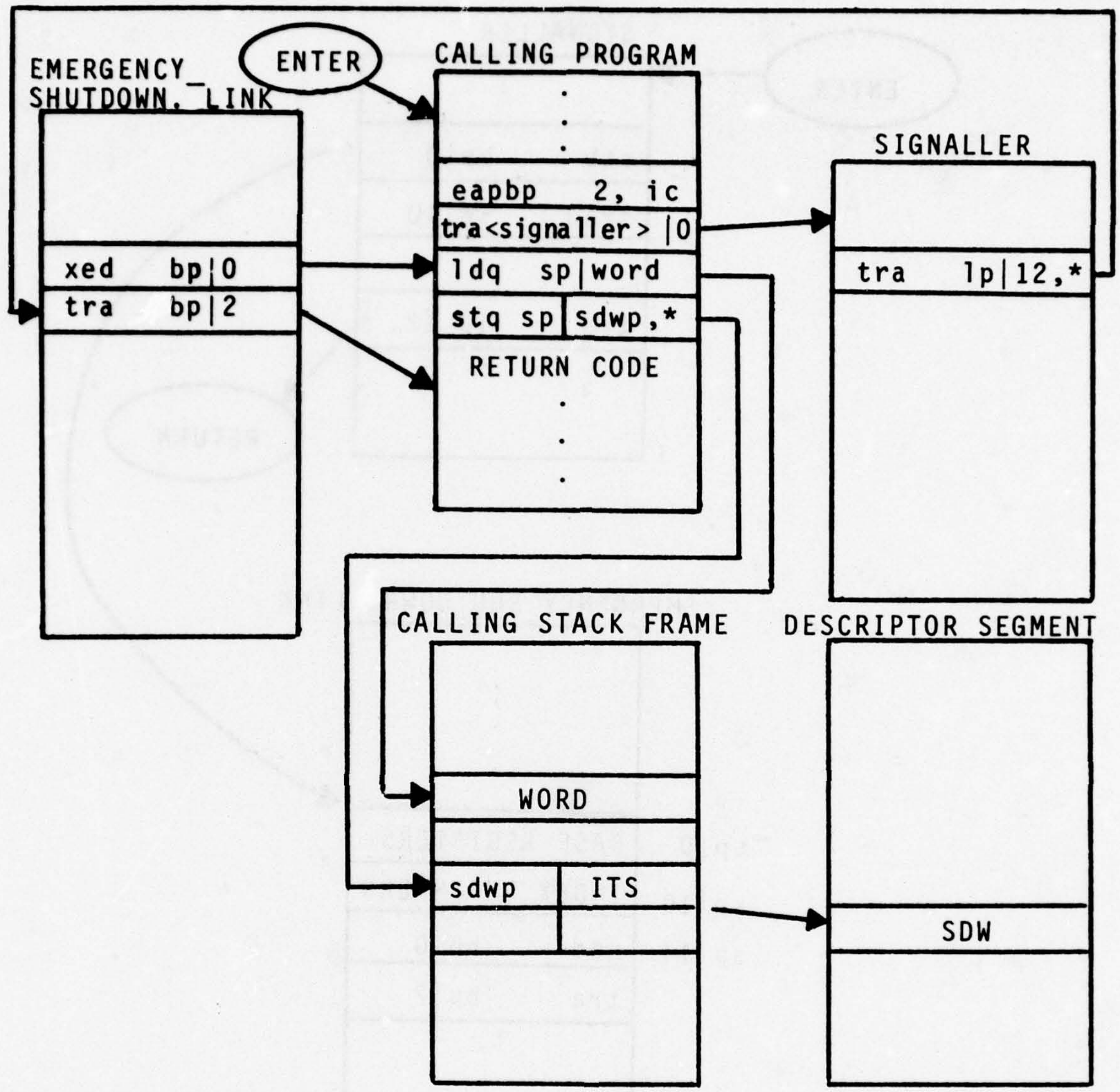


Figure 13 UNLOCKED STACK BASE (STEP 2)

approximately 2 hours. It is believed that if the attacker were properly trained, that RSX-11D can be penetrated in a few minutes.

The operating system RSX-11D for the PDP-11 series has the capability of providing password protection for a given user's data directory and/or individual files. The objective of the attack is to obtain the password of a file unknown to the user. Two methods of doing this are:

Method 1

(a) Gain physical access to the machine control panel and an active teletype

(b) Log in on user directory.

(c) If user directory is password protected, enter a password but do not hit carriage return.

(d) Halt machine.

(e) Enter odd number into Register.

(f) Press Continue.

(g) Take core dump via dectape or magtape.

(h) Run core dump analysis program.

(i) Look for Task . . . HEL the login program.

(j) Observe the needed password approximately 100 octal locations from the beginning of the task in standard ASCII encoding.

Method 2

(a) Gain access to any terminal for the system.

(b) Log in as any user number you know.

(c) Run PIP (Peripheral Interchange Program).

(d) Set default UIC to (11, 1).

(e) Copy task . . . HEL into your user area.

(f) Run ZAP program.

(g) Make your copy of . . . HEL fixable in core by changing task header.

(h) Run . . . HEL with Fixed in core option.

(i) Type in user number of password protected user directory.

(j) Use OPEN system directive to look into task . . . HEL at about 120 octal locations from the beginning of the task and observe the *required password in standard* ASCII.

3.6 Conclusions on the Threat. There has been no attempt in this chapter to exhaustively present criminal tactics. Rather, the intent is to give examples that have been effectively used in practice. These examples are intended to show the reasonableness of the ease with which a malicious user can penetrate the IAS.

Despite the limited number of examples shown, it must be concluded that there is *no* known way of preventing the malicious user from gaining command of the system. It has been shown that denying programming access will not work. The example of the data base query trapdoor illustrates that point. Further, there are undoubtedly combinations of hardware, software and procedural flaws that would achieve the same result.

We are forced to conclude that the malicious user can easily gain command of the system. The emphasis in the next chapter will be on examining the effectiveness of various architectural variations in view of this vulnerability.

SECTION 4.0

EVALUATION OF CANDIDATE APPROACHES

4.0 EVALUATION OF CANDIDATE APPROACHES

From the presentation in the preceding chapters, the problem of access control continues. Assume that the data base system shown in Figure 8 contains two types of variable length records stored on disk and has two types of users. The records are either Secret or Top Secret. Correspondingly, the user's privilege is either Secret or Top Secret. A user with Top Secret clearance can access any record in the data base, but the user with Secret clearance is denied access to Top Secret records. The problem is to prevent the malicious user with Secret privilege from getting Top Secret records.

The problem has been reduced to its essential elements. Obviously, if there were N compartments comprised of Unclassified, Secret, Top Secret, special intelligence access and "need-to-know" categories, the problem of restricting access would be similar, only more complex.

The problem of penetrating two or more computers is not really relevant. If the malicious user can gain access to a single computer in a matter of minutes, command of two or more computers can be accomplished in a brief period of time. While multiple configurations add some complexity, they do not change the essential problem.

The examination of alternative architectures is centered on the employment of external control mechanisms, as prescribed in the study's Statement of Work. The external control devices are sometimes referred to simply as "guards" or more commonly in the literature as "reference monitors." The guard terminology is more descriptive. The function of the reference monitor is analogous to that of a guard in a library containing secret and top secret information. The user provides identification to the guard and after the identification has been verified, the guard references the requestor's access list to find the user identification and privileges, that is, either Secret or Secret and Top Secret. The user requests a record and the guard checks to see that the classification of the record is authorized to that particular user, and either grants or denies access, depending on that determination.

The following paragraphs present a review of the deficiencies or vulnerabilities of the IAS as presented in the preceding chapter.

4.1 <u>Existing IAS</u>. As shown in the preceding chapter, the vulnerabilities of the AN/GYQ-21(V) fall into three categories:

- The criminal gains command of the system, that is, the ability to write anywhere in memory, dump single words or blocks, and execute arbitrary programs. This does not in itself accomplish espionage because the protected records of the intelligence system are stored in disk file, but it is a prerequisite first step.

- Given the ability to command the operating system or write programs that will dump the data base, the criminal proceeds to obtain a Top Secret record using a secret privilege.
- Alternatively, the criminal can attack the reference monitor or guard, that is, the mechanism in the IAS that guards access privilege to records, given the computer's understanding of identity of the requester and the access privileges.

Since gaining command of the system is a prerequisite to the other vulnerabilities, if modifications can be made to prevent this from happening, the IAS can be made impenetrable. What are these modifications? The first method is to eliminate the possibility of trapdoors or other flaws that can be exploited. Unfortunately, Harris does not know how to accomplish that objective. Testing for trapdoors and other flaws would be impossible.

The second method of preventing the criminal from gaining command of the system would be to build an effective reference monitor, that is, entrance guard that would prevent the malicious user from entering any unauthorized commands into the system.

4.2 Entrance Guards. An entrance guard is a reference monitor. It examines each request of the user and grants or denies the request, depending upon the privileges of that user. The reference monitor may be made impenetrable to attack by building it as an external device, totally isolated from the main computer system. The software in the reference monitor is small enough that it can be certified, thereby preventing it from being tampered with. Unfortunately, there is no known way to prevent the malicious user from bypassing the entrance reference monitor.

To see how a malicious user may bypass an entrance guard, it is first necessary to review the command language of the terminal. This varies from application to application, but Chapter 5 develops a language that may be used as an example, even though it is not typical. The commands are as follows: DISPLAY, PRINT, DELETE, ADD, and CHANGE. The ADD and CHANGE commands have the following structure:

ADD (report), NAME = nnnnn, DATA = ddddd.

The problem is the entrance guard has no way of telling whether or not the DATA hides a malicious command. Consider, for example, that the trapdoor could be triggered by entering QZKX, followed by a sequence of malicious commands outlined in the preceding section; READ, (address), WRITE, (address) (number), GO TO, (address), END.

Since the form of the trapdoor trigger and the commands are unknown to the reference monitor, it cannot detect them and, therefore, prevent them from being executed. It is conceivable that the user could be denied access to all commands except one; for example, the PRINT command,

reserving the data base modification commands to other terminals of higher privilege. Still, the malicious commands could be hidden in the PRINT format. It would be awkward, but far from impossible.

We are forced to conclude that entrance guards are useless as long as there is the possibility of trapdoors or other flaws inherent in the software. Therefore, we must assume the malicious user can gain full command of the system and explore methods of making the IAS impenetrable under this assumption.

4.3 Protected Access Controls. A necessary, but insufficient condition for an impenetrable IAS, (assuming the criminal can fully command the system) requires that the access control mechanism be protected from penetration attempts. Failure to protect the access control mechanism can result in falsification of user identification and access privileges. It is feasible to build an isolated, external access control mechanism that is secure from attempted tampering by a criminal.

The procedures for establishing user access privileges vary with different intelligence systems but generally follow this sequence:

- The user types in on the terminal a request for service. The sign-on includes user identification, charge number and other pertinent data.
- The computer commands the user to verify identification usually by requesting a password. The intelligence community has experimented with other means of identity verification including hand geometry,voice prints and fingerprints. However, an examination of the vulnerability of masquerading attempts is beyond the scope of the current contract. It will be assumed throughout, a simple password is adequate to verify claim identity.
- At this point in the sign-in dialogue, the computer is satisfied with the user identification and proceeds to consult the access list to determine user privileges, that is, either Top Secret or Secret.
- Finally, a computer system passes these access privileges to the reference monitor responsible for controlling access of that user.

In all intelligence centers, the sequence of sign-on control is physically contained within the IAS or large general purpose computer connected through the IAS. Obviously, if the criminal can command the IAS or the large computer system, (which must be assumed), then the user can freely change identity or access privileges.

Figure 14 shows a system to prevent this from occuring. The access control mechanism comprised of an access control center and a guard for each terminal is completely external to the IAS. In practice, the access control center would be a small minicomputer or a microprocessor. The

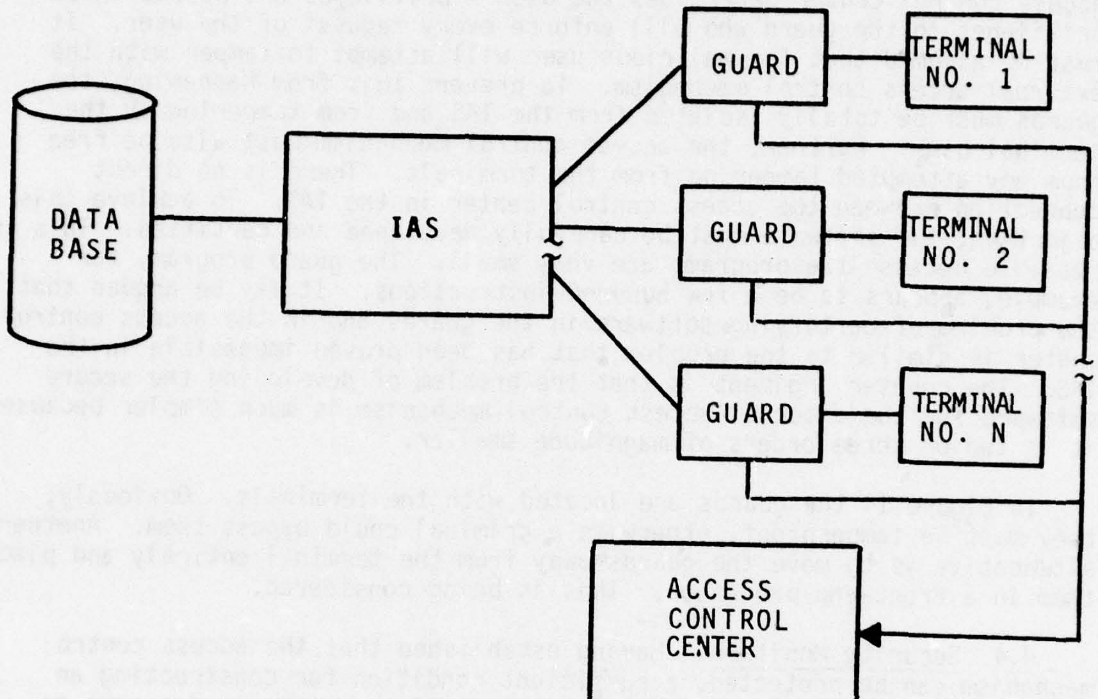


Figure 14 EXTERNAL, ISOLATED ACCESS CONTROL MECHANISM

guards would be microprocessors; in the feasibility model they are Digital Equipment Corp.'s LSI 11's. When the user signs on the terminal, the access control center determines the user's privileges and passes those privileges to the guard who will enforce every request of the user. It must be assumed that the malicious user will attempt to tamper with the external access control mechanism. To prevent this from happening, the guards must be totally isolated from the IAS and from tampering by the terminal user. Further, the access control mechanism must also be free from any attempted tampering from the terminals. There is no direct connection between the access control center in the IAS. To achieve this objective, the software must be carefully developed and certified. This is feasible because the programs are very small. The guard program, for example, appears to be a few hundred instructions. It may be argued that the problem of certifying software in the guards and in the access control center is similar to the problem that has been proven impossible in the IAS. The counter argument is that the problem of developing the secure software for the external access control mechanism is much simpler because it is two or three orders of magnitude smaller.

In Figure 14 the guards are located with the terminals. Obviously, they must be tamperproof, otherwise a criminal could bypass them. Another alternative is to move the guards away from the terminal entirely and place them in a front-end processor. This is being considered.

4.4 Security Monitors. Having established that the access control mechanism can be protected, a sufficient condition for constructing an impenetrable IAS would be a method of preventing unauthorized access to records, even though the criminal had command of the IAS. One method of accomplishing this is through the use of security monitors. Security monitors are an effective means to deter attempted espionage and are in use in the National Military Intelligence Center, the Defense Intelligence Agency On-Line System, and other intelligence systems. The theory of security monitors is to detect that a breach of security has occurred. This knowledge is often sufficient in itself, for it allows the criminal to be caught, keys to be changed, and generally to take measures to neutralize the negative effects of the espionage.

A problem with security monitors is the criminal can readily falsify the logs and erase the evidence. For specific examples of how this is accomplished, refer to the document referenced in Footnote 4, Paragraph 3.0.

4.5 Isolated Security Monitors. Just as it is possible to isolate and protect the access control mechanism, so it is feasible to isolate the security monitor, and prevent the malicious user from eras ng the evidence. If erasure were the only problem associated with security monitors, that would be so. Unfortunately, it is not. They can be spoofed in the same way as the entrance guard. Suppose the user types in the sequence:

ADD (report), NAME = nnnnn, DATA = ddddd.

The data field may contain several hundred characters. Suppose the data field contains a trapdoor trigger and several commands to read and write privileged data in the IAS. The log believes the data item is a normal addition of a report to the file and has no way of detecting that it is, in fact, a malicious command sequence.

4.6 Data Base Guard. A method of denying user access to unauthorized records, even though the user commands the computer, has been proposed by Bob McGill of Harris. It is known as the Data Base Guard Approach.

Figure 15 illustrates the approach. The data base shown on the extreme left-hand side of the figure contains two types of variable length records, Secret and Top Secret. A multiported memory filters these records. IAS No. 1 is permitted all records in the data base, both Secret and Top Secret. IAS No. 2 is permitted Secret records only. IAS No. 2 may request a Top Secret record. The data base will be searched, but when the record comes to the multiported memory, the classification field will be sensed and the request aborted. The bottom of Figure 15 shows the organization of the record preceded by the classification field. The access control mechanism connects a user to one of the two processors, depending upon access privilege at sign-on. The engineering problems associated with this approach are threefold:

- Design and certification of the multiported memory - The multiported memory consists of a small microprocessor, hardware, and a brief program. An obvious penetration tactic would be to change the software or the hardware so as to allow Top Secret information to be communicated to the secret IAS. To prevent this from happening, design must be carefully executed and the software must be certified.
- Design of the access control mechanism - Obviously if a user with Secret privilege is switched to the Top Secret computer, all control is circumvented. The access controls would consist of an access control center which verifies the identity of the user and obtained user access privileges from a recorded list. It would then perform a switching function to the proper computer. This appears to be no serious problem and the design is not pursued further in this report since it appears to be straightforward.
- Cost - An IAS must be dedicated for each compartment. In the case illustrated in Figure 15 this is not a serious problem because there are only two compartments. In practice, however, there may be a dozen or more requiring that a large number of computers be purchased. The increase in cost is not as significant as might be expected. The central processing units are relatively inexpensive by comparison with the cost of peripheral equipment.

Harris believes the data base guard approach will prevent a malicious user from gaining unauthorized information. On the basis of this, we find that it is feasible to develop a secure data base system, although at

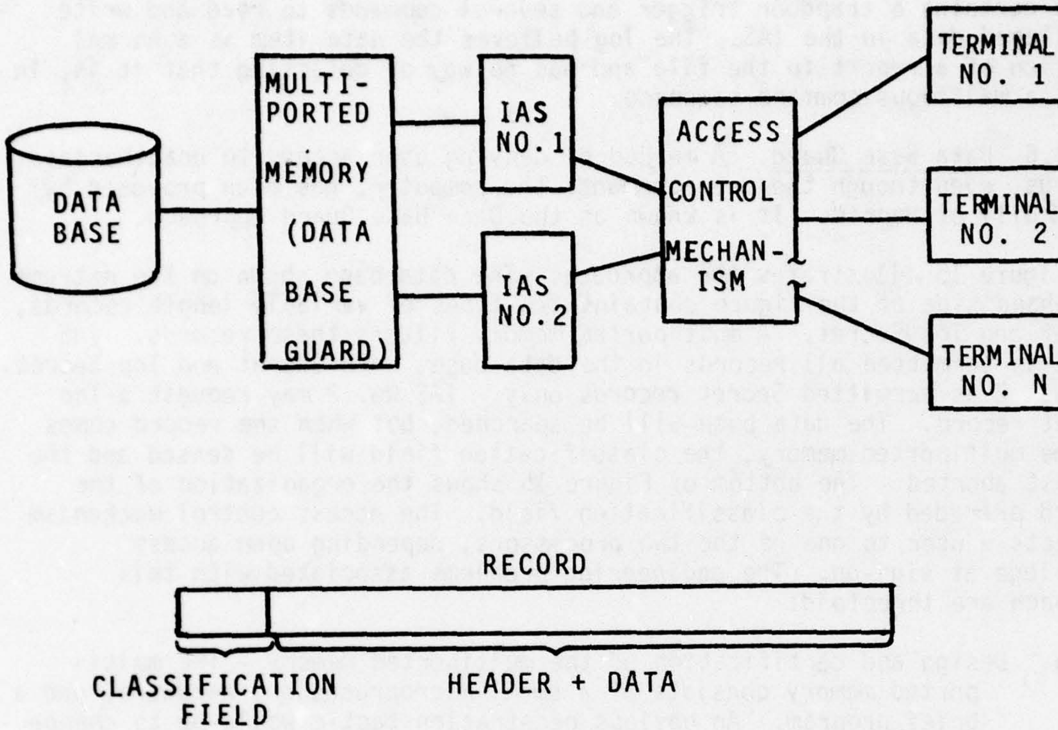


Figure 15 DATA BASE GUARD APPROACH

substantial additional cost. The search will continue for alternative approaches that are just as effective, but would incur less cost.

4.7 Tag Approach. A second method of securing records in the IAS has been proposed by Bob McGill of Harris. Suppose every record in the data base contained an 8-byte tag as shown in Figure 16 and the external access control mechanism operates as shown in Figure 14. The system operation would follow this sequence of events:

- The user begins the sign-on procedure and requests access to the data base system.
- Referring to Figure 14, the user input is recognized by the guard as the beginning of a sign-on procedure and control is passed to the access control center (ACC).
- The ACC accepts the request of the user which includes the user's Social Security number for identification and requests verification of the User's identity.
- The user types in the password.
- The ACC now accepts the user identified by Social Security number as a valid user.
- The ACC checks the access list colocated with the ACC. The list is comprised of Social Security numbers and the access privilege of each user (either Secret or Top Secret). Assume the current user has Secret privilege only.
- The ACC commands the guard to pass only Secret records to the user. Control is now passed to the user for communication via interactive dialogue with the data base system.
- The guard is totally transparent as the user queries the data base.
- A record is returned. The guard is able to discriminate between the various types of messages communicated from the IAS to the user and is able to recognize a record.
- The guard strips off the record tag to determine whether the record is Secret or Top Secret. If the record is Secret, the guard allows it to be displayed on the User Terminal. If the record is Top Secret, the guard does not communicate the record to the user but rather informs the access control center of a breach in security.
- After the classification has been established and before the record is communicated to the user, the guard checks for errors in the tag and errors in the message, utilizing the parity information as shown in Figure 16.

In evaluating the effectiveness of the tag approach it is first necessary to postulate the threat. Consider the threat of accidental disclosure. In that event, the 8-byte tag shown in Figure 16 is something of an overkill. All that is needed is a bit pattern to give the classification of the record and a field that identifies the record. It may also be desirable to include one or more parity bits to confirm that the security level is correct.

Considering the threat of malicious attack, the requirements on the tag approach are much more severe. First of all, if the tag is left in the clear, the criminal can be expected to falsify it, for example, changing the Top Secret classification to a Secret classification. Accordingly, the tag must be enciphered to prevent forgery. That is why the tag is chosen to be 8 bytes long; that is, the length of the block cipher used in the National Bureau of Standard Encryption Algorithm.

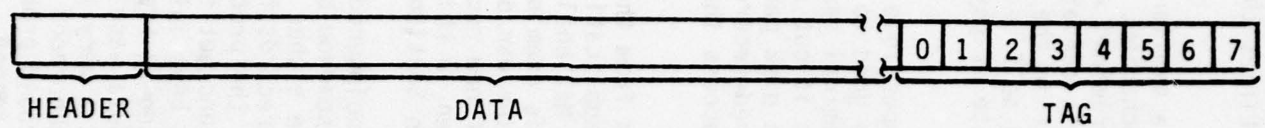
Even if a tag is scrambled, that in itself is not sufficient to prevent forgery. If the tag contains no parity bits or other record unique identifiers, the criminal can simply lift the tag from the Secret record and replace it on a Top Secret record. The puspose of the record parity bytes is twofold. It makes the tag unique and it provides a basis for determining whether or not the record is in error.

It is our opinion that the tag can be so designed as to be impossible to forge. Since the exit guard is so designed as to prevent the user from getting the requested record if the tag has been removed, mangled or otherwise fails to pass parity check, the user is denied unauthorized access; however, there remains the potential for bypassing the exit guard.

In a tag approach, it must be assumed the criminal commands a system and will try to get the record out past the guard any way possible. This situation is analogous to that of a burglar who has found the critical information, but there are exit guards posted at the exits that will inspect the document. Obviously, the burglar will try to get the information by the guard any way he can, by mailing it to himself, signalling out the window, throwing the information out the window placing it in trash collection barrels, etc. Figures 3 through 7 show the potential exits in NMIC. There are many tapes, disk packs, line printers and terminals which, if left unguarded, provide the potential for the malicious user to get the information out, including copying files and creating arbitrary messages. Accordingly, an effective exit guard system must guarantee there are no loopholes through which the malicious user can exit the information.

One way the malicious user would try to bypass the guard would be to disguise the classified information as an unclassified administrative message. One example is, if an operator types in an error, the system communicates back to the operator that error has been made. The error message could contain classified information. Unless the guard is able to distinguish between records and administrative traffic, the whole system will fail.

4.8 Enciphered Records. Suppose the data base were comprised of variable length records, each an integer number of 8-byte words (this is no problem in the IAS as the standard for disk retrieval is a sector length which is 256 words or 64, 8-byte words). A record requested by the terminal user is intercepted by the guard. There would be a clear text field identifying the record as such. The guard deciphers the record, reads the classification field, and matches the user's access privilege



TAG BYTES: 0 = RECORD CLASSIFICATION, TS OR S

1,2 = TAG PARITY, DETECTS ERROR IN TAG

3-7 = RECORD PARITY, DETECTS ERROR IN RECORD

Figure 16 RECORD TAG

against the classification of the record. If they match, the guard communicates the record to the user. If a Secret user has requested a Top Secret record, the guard destroys the record and notifies the access control center.

The initialization of the guard takes place in a similar manner to that of the tag approach, using an external, isolated mechanism. There are several schemes for passing the key. Obviously, the criminal will try to steal the key, and if successful, the whole system is defeated. One simple method of protecting the key is to manually place it in the guard and design the guard so that it is both tamperproof and self-destructs; that is, it is designed so that anyone trying to penetrate it physically erases the key from memory.

Because the records are enciphered and it is impossible to extract the clear text without the key, it does the criminal no good to bypass the guard. As in the case of the tag approach, the criminal can try to disguise the record as an administrative record, print it out on a remote printer, copy it on a disk pack, and then steal the disk pack, etc. For this reason, it is Harris' belief that the enciphered record approach is potentially a more effective means of controlling access than the tag approach.

The disadvantage of the enciphered record comes from the fact that the records cannot be processed. In all intelligence applications, some processing is required of the data base records. In the National Military Intelligence Center, for example, each incoming record is examined to determine to message addresses. Obviously, such an examination cannot take place on the enciphered record. This raises the question of how records can be processed when they are enciphered. The approach taken in this report is that of RED-BLACK multiprocessing described in detail in Section 6.0.

There are other problems associated with the enciphered record approach. First and most important is that all records must somehow be enciphered when they first enter the data base. This would require either an on-line device or a special batch processing step to prepare the records for the data base. This can add expense to the system. Then there is the problem of changing keys. If it is decided that the keys cannot be found out for a period of months or years, it is probably adequate to change keys infrequently, but if the Government determines the keys should be changed at frequent intervals, such as daily, this poses a severe problem. The data base would have to be read out and deciphered and reenciphered. This is a very slow process. Random access to a track in the IAS is in the neighborhood of 40 milliseconds. Reading and writing a record on the track could probably take place at speeds of not much greater than 20 records per second. Some of the intelligence data bases have several million records. Accordingly, the time to change keys could be very long.

4.9 Other Approaches. The architectural variations to the IAS to achieve information protection presented thus far are not exhaustive. There

are other approaches to prevent the unauthorized user from gaining access to records or removing records from within the IAS. Three additional approaches are briefly sketched below.

- Secure Software - If it were possible to develop software for the IAS that would be free of trapdoors and other flaws, then the malicious user would be denied the ability to command the system and thereby unable to gain unauthorized access. Consideration of a secure software approach is not within the scope of this study.
- Protected Index - A method that has been proposed to keep the malicious user from gaining unauthorized access to records is to hide the index. In the intelligence applications, the data base utilizes an inverted file structure. The theory goes that if the user cannot access the index, the record address is not available and access is denied. The weakness of this approach is twofold; first of all, the problem of preventing the malicious user from accessing the index, even if this can be achieved the user can still browse through the records and gain access to Top Secret records without authorization.
- Combinations of the Approaches - It may be possible to combine approaches such as the entrance guard and the tag approach, in order to achieve a high degree of protection. While the entrance guard is not foolproof, the tag approach has certain vulnerabilities. By combining them, one method may offset the weaknesses of the other. This approach of combining methods may have merit. It has not been pursued further in this report because other methods are viewed to be more promising.

4.10 Preliminary Findings. Of the ten approaches to achieving data base security in the IAS, seven were found insufficiently promising to merit further consideration. The remaining three ranked in order of their promise to provide the basis for an impenetrable IAS are as follows:

- Data Base Guard - It must be anticipated that a computer criminal would attempt to attack the access control mechanism that switches the user to the proper machine initially, and then attempt to attack the multiported controller that disallows Top Secret records from being read into Secret IAS. Of these vulnerabilities, switching mechanism seems to be a straightforward design and would offer little that the malicious user could attack. The multiported controller design is examined in detail in Section 7.
- Enciphered Record - The points of attack or potential vulnerabilities of this approach are the RED-BLACK multiprocessing, stealing of the key, tricking the exit guard into deciphering the data and cracking the code. Of these vulnerabilities, the RED-BLACK isolation is critical and is examined in Section 6. It is impossible to speculate on the other vulnerabilities without more detailed design information. Section 5 presents the design of a feasibility model to execute the enciphered record approach.
- Tag Approach - Potential vulnerabilities from this approach come from the potential for the malicious user to bypass the exit

guard by typing out the Top Secret record on other terminals, writing them on magnetic tape or disk pack and then stealing them or disguising them as administrative messages. The second potential for defeating the approach comes from forging the tag. The latter is not considered as serious as the former. Section 5 presents the feasibility model that can be utilized to design the tag approach and discusses the problem in more detail.

The latter two approaches assume the availability of a block encryption encipher/decipher device. The National Bureau of Standards 64-bit block encryption device may be utilized for non-Military applications. It is assumed that the Government will furnish a block encryption device suitable for Military applications. Because of this assumption, the potential of cracking codes to obtain enciphered records or falsify a tag are considered unlikely. For purposes of experimentation, a modified penetration evaluation requiring the use of a 128-bit key will be considered. Consideration will also be given to use of the Duffie Hellmen algorithm.

SECTION 5.0

AN EXPERIMENTAL, ENCIPHERED DATA BASE SYSTEM

5.0 AN EXPERIMENTAL, ENCIPHERED DATA BASE SYSTEM

The evaluation of candidate approaches presented in the preceding section established that three approaches have promise in building a secure data base system. Section 7 will present one of these approaches, the data base guard approach. It is considered less important than the other two because it is a brute force approach and requires the allocation of one computer central processing unit to each security compartment. This section discusses the other two approaches, the enciphered record, and the enciphered tag.

Since the initial objective of this study was to determine if the AN/GYQ-21V could be made secure for intelligence applications, the analysis of this section will establish that it is possible that either or both of the approaches will achieve the objective. In establishing feasibility, there are basically two approaches. The first is a theoretical analysis involving the design of critical characteristics versus cost. The other approach is to build the system or a portion thereof and see if it works. We believe that the experimental approach is superior in this instance to the analytical approach and, therefore, is used to examine feasibility.

5.1 Background. Before proceeding with the description of the feasibility model that incorporates elements of both the enciphered record and the enciphered tag approach, an overview will be given of this section.

5.1.1 Objectives. There are three objectives in developing the feasibility model:

- Develop a secure data base system. The primary objective is to establish that it is in fact possible by any method at all to prevent espionage in an intelligence data base system using the interactive analyst station.
- Quantify the relationship between the principal performance measure (time required to gain unauthorized access) and cost. It is not enough to prove academic feasibility but engineering feasibility must also be shown in this regard. The solution must be affordable and cost, or engineering design, becomes a prime consideration.
- Quantify the relationship between the other two performance measures (delay and reliability) and cost. The introduction of hardware for the external access control mechanisms will introduce delays in the processing of data base queries and other transactions. Further, the additional hardware introduced into the system will be a source of unreliability in operation. The extent to which performance is degraded will be explored as part of the study.

5.1.2 Method. Due to the limitations of funds, personnel and time, the analysis must be limited to those areas that will have the largest payoff. Primary emphasis will be placed upon the enciphered record approach. The problem of proving isolation between the red and black processors will be demonstrated in Section 7. The enciphered record approach has the engineering elegance of reducing access by encrypting the

entire record and is a method which has been demonstrated in telecommunications for decades to be an effective means of denying enemy access to information, even with free access to the encrypted data.

The first emphasis in the effort will go to the development of a feasibility model. Following this, at some later time (FY 78) the design and development of an engineering model will begin.

5.1.3 Message Flow in the Model. The model to be developed is representative of the types of data base systems encountered in the National Military Intelligence Center, and other intelligence applications. The flow of information can be visualized in five parts as follows:

- Operator enters transaction into the LSI-11 (guard). The operator requesting information or modifying the data base utilizes a terminal to communicate with the computing system. The terminal in turn is controlled by an exit guard in the case of the model simulated to be an LSI-11. The operator desires to execute certain transactions, such as QUERY THE DATA BASE, DELETE RECORDS, CHANGE or CREATE RECORDS. It is the function of the LSI-11, the exit guard, to control all of these transactions so as to achieve compartmentalized security.
- LSI-11 passes the transaction (query) to the 11/45. The exit guard has limited computing capacity and it is desirable that the software size be limited so that it may be certified. Accordingly, the processing is done in the main machine, in the case of the model being developed it is a PDP-11/45.
- PDP-11/45 returns all records that meet the search criteria to the LSI-11. The query requests all records that meet the search criteria, such as name, location, time, etc. The central processor searches the records and produces all those that meet the search criteria.
- The LSI-11 matches the record compartment against the user privilege. Two methods are implemented in the feasibility model, the tag method and the enciphered record. In the case of the tag method, the LSI-11 computes the parity on the enciphered record and compares parity with that contained in the tag. It further deciphers the compartment identification of the tag and matches that against the user access privileges. In the case of the enciphered record, the record is deciphered only if the user is authorized access to that record.
- If the user is privileged to that compartment, the records are printed. In either approach, if the user access privilege recorded in the guard matches that of the record that has been requested, the records are printed on the teletype printer.

5.1.4 Problems Examined by the Feasibility Model. There is a limitation of time and personnel available but it is desirable that the following performance measures of the feasibility model be examined and analyzed.

- Securing the number of responses to a data base query. In many intelligence applications, it is not enough to simply hide the records in the data base to prevent an unauthorized user from getting them, but it is also necessary to prevent an unauthorized user from finding out how many hits there are. For example, if a user was unauthorized to know the location of ships in a particular ocean, it might be possible for him to query the data base and find the number of ships in the Indian Ocean, thereby giving a certain amount of information.
  The method of achieving denial of the number of hits to an unauthorized user in the feasibility model is to return all records that meet the search criteria to the guard, and upon request, print only the number of hits that the user has access privilege for; that is, the access rights of the user will be matched against each returned record and only those records that are authorized will be included in a tally of the number of hits.
- Transactions other than query. The primary emphasis in the feasibility model study will be upon query of the data base, based upon a number of keys and an "and" condition among those keys. There are, however, certain other transactions that must be studied. They are: DELETE RECORD, APPEND RECORD, CHANGE RECORD, CREATE RECORD. The method of handling these transactions is identical to the query transaction; that is, the user will be permitted to execute those transactions only if authorized.
- Administrative traffic. The operation of a terminal position requires that certain messages be exchanged with the computing system in addition to those of data base transactions. They might be, for example, communication of a message from one operator to another, messages on the status of the equipment, the number of records in a queue, etc. All messages must be controlled by the exit guard, otherwise the system may be defeated. For example, a malicious user could search the data base, get the information required, disguise it as an administrative message, and get it by the exit guard. To prevent this, all records must be tagged, administrative as well as data base operations.
- Enciphering algorithm performance. The most obvious way to defeat the security measures of the data base system is a direct assault upon the enciphering algorithm. The feasibility model will use a simple scramble algorithm which has little defense against a sophisticated agent. No attempt is made in the model to go to more sophisticated approaches. Although, as time permits, the National Bureau of Standards Encryption Algorithm will be explored as well as certain encryption algorithms based upon the principles of substitution and permutation sequence, and the Diffie Hellman algorithm. There is no attempt on the part of Harris to go into this problem area which is admittedly central to the overall performance of the data base scheme, but rather to obtain help from the Government. The design and development of enciphering algorithms is the sole province of the Government and it would not be wise to duplicate an extensive capability. Rather, we hope to obtain assistance on this matter.

- Sabotage prevention, detection and recovery. The design of the guard system is intended to detect deliberate efforts to falsify the tag or the record itself. This function is accomplished by the use of error detecting codes. When errors are detected, they are referred to the access control center, who examines the frequency of errors and initiates certain recovery procedures and alarms when they exceed an acceptable value.
- Espionage threat analysis. The design of the feasibility model is a vehicle for testing vulnerability. Certain threats will be postulated, including tampering with the guard, attempting to disable it, bypass it, or otherwise circumvent its operation. Additionally, certain privileged users, such as maintenance personnel, will be examined to determine whether the access controls can be defeated by somehow compromising these privileged users.

5.1.5 Summary. The remaining portion of the presentation of this chapter falls into five sections as follows:

Paragraph Number	Title
5.2	LSI-11 (Guard) Top-Level Design
5.3	11/45 Top-Level Flow
5.4	Query Processing
5.5	Data Base Query Operations
5.6	Record Tagging and Detagging

5.2 LSI-11 Top Level Flow. The experimental model should be as close as possible to the ultimate operational system. Accordingly, the central processor used in a PDP-11/45, which is software compatible with the AN/GYQ-21 (V). Preliminary sizing of the exit guard places it in the microprocessor category. For a first cut, the LSI-11 is used.

5.2.1 Exit Guard Design Approach. Figure 17 shows the equipment configuration used in the feasibility model. There are three principal parts. The exit guard and terminals, which control and initiate access, the access control center, and the central processing unit (11/45) used for storage and retrieval. Communication between the operators and the central processing unit, that is, the LSI-11's and the 11/45's, is via shared memory. There are three interactive analyst stations (IAS) simulated in the model. IAS No. 1 is an LA-36 local terminal used for entering and retrieving information in the central processing unit. IAS No.'s 2 and 3 have controlled access through the exit guard via the shared memory to the central processing unit.

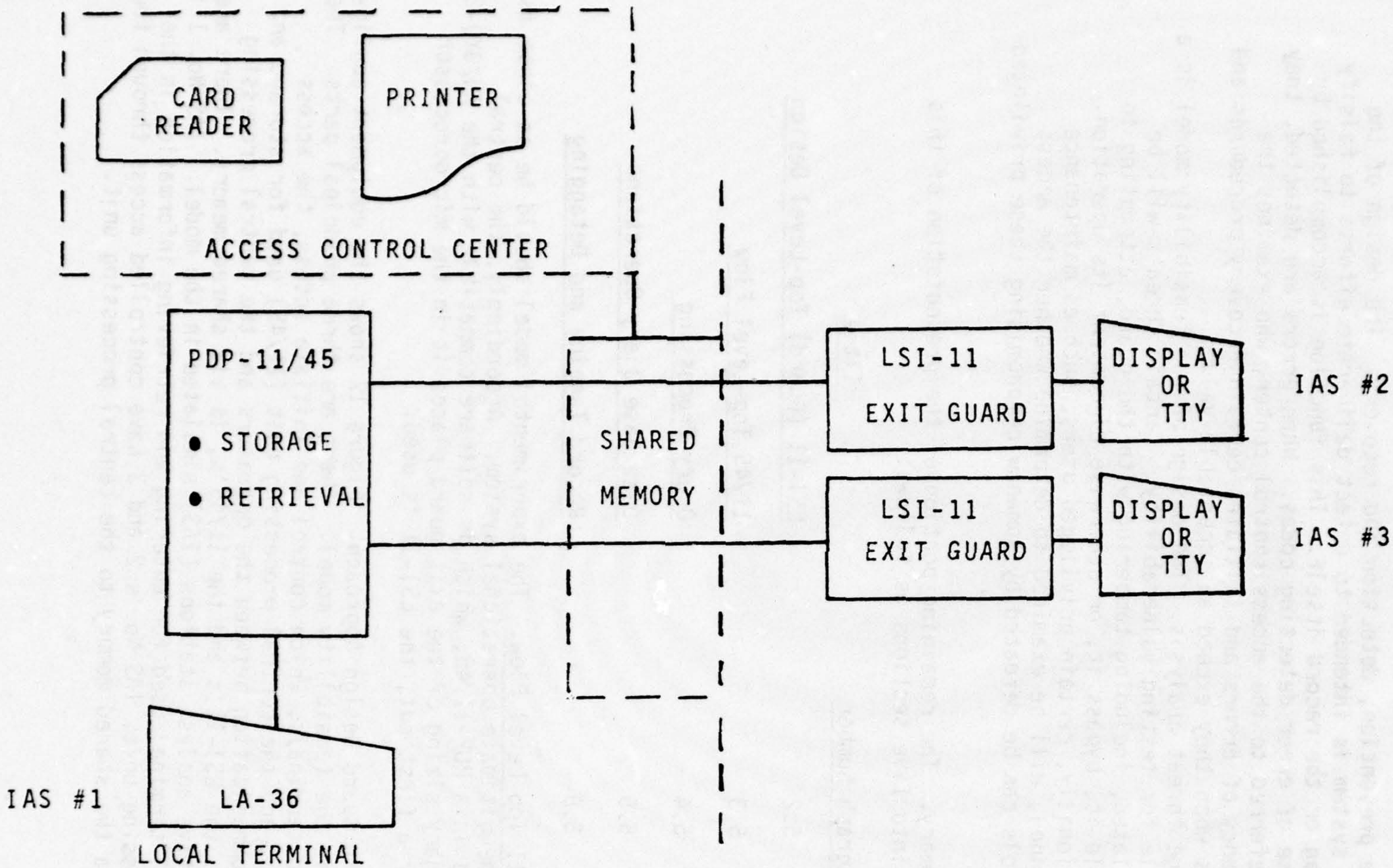


Figure 17
EXIT GUARD DESIGN APPROACH USING HESD'S COMPUTER ARCHITECTURE DEVELOPMENT FACILITY

The access control center is, as the name implies, the central control for all of the exit guards. In the experimental configuration, it is comprised of a cardreader and printer. It passes the keys to the exit guards. The exit guards transfer control to the access control center when an anomaly occurs. In the operational system, the access control center will determine whether these anomalies are just accidental errors in the system or deliberate attempts on the part of a malicious user to penetrate the data base.

5.2.2 LSI-11 Top Level Flow. Figure 18 shows the overall program operation in the LSI-11. It is divided into two parts, initialization and operation.

The exit guard reads inputs from the keyboard. That is, it receives the characters and assembles them into messages. The carriage return is used by the operator to indicate a completed entry. Next, it processes the entry and communicates the results of the operator's input, that is, the assembled message, to the 11/45.

After the input message has been communicated to the 11/45, it awaits results. The results are in the form of a number of records which meet search criteria. When results are inputted from the 11/45, they are verified by the exit guard. This flow diagram utilizes the enciphered tag approach. Here the tag is verified; that is, the compartment contained in the tag is matched against the access privileges of the user. If the tag passes the test, the program branches to the good tag branch. If not, it branches to the bad tag branch.

If the tag is good, the record is communicated to the operator by printing or displaying it. If the tag is bad, a message is sent to the access control center and the records are not communicated to the operator. At the completion of each transaction, control is returned to the read keyboard module.

5.2.3 Initialization of the LSI-11. Table 3 shows the operation, purpose, inputs, and processing of the overall operation entitled, "Initialize the LSI-11." There are no outputs. The purpose of

Table 3. Initialize LSI

Operation:	Initialize LSI
Purpose:	To initialize the LSI-11 software and hardware prior to system operation
Inputs:	• Initial program load • Key for tagging and detagging record tags
Processing:	• Set up tables for tagging and detagging • Clear initial flags • Start keyboard inputs • Set up communications with 11/45 • Set up COMPOOL initial conditions

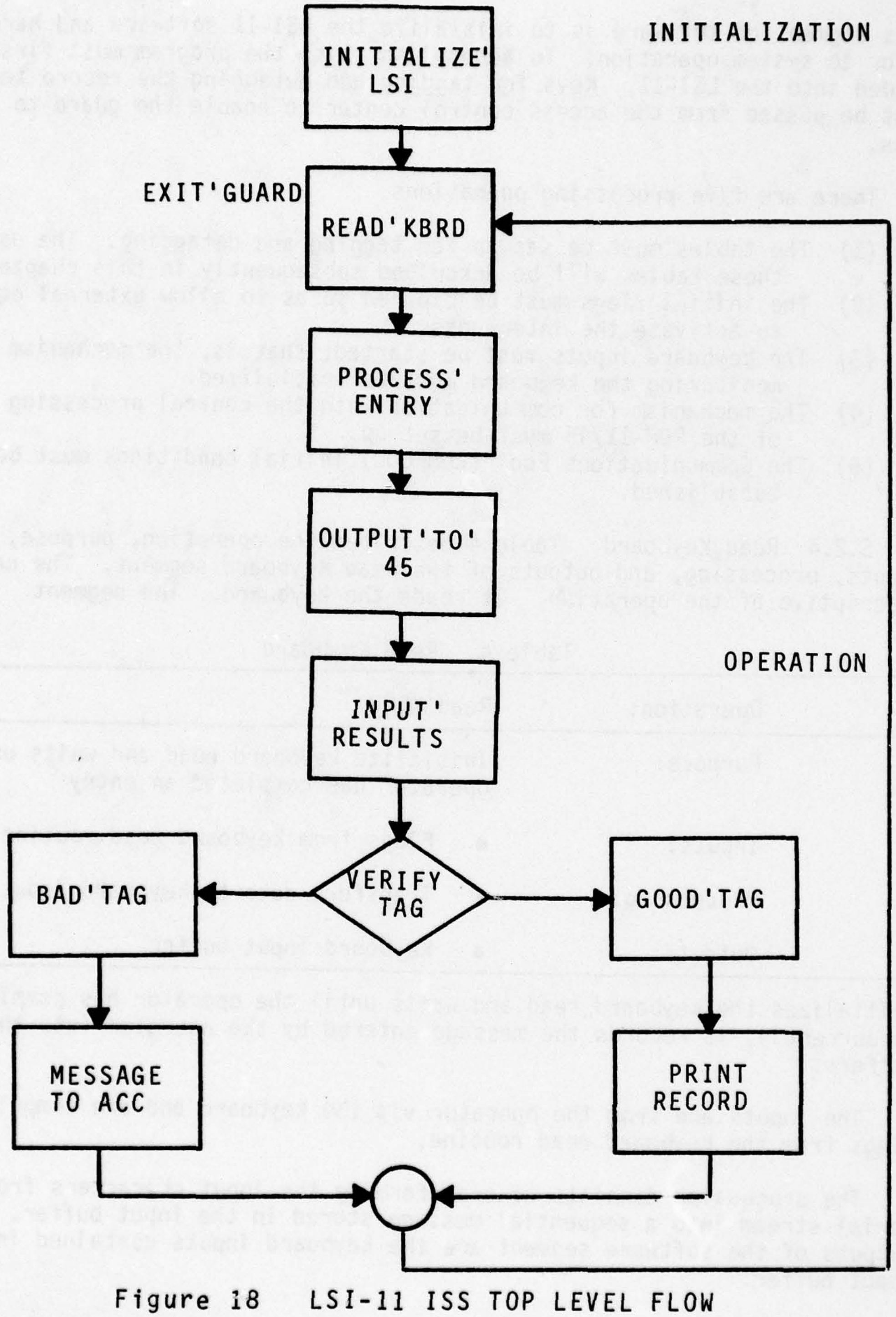


Figure 18 LSI-11 ISS TOP LEVEL FLOW

this segment of software is to initialize the LSI-11 software and hardware prior to system operation. To accomplish this, the program must first be loaded into the LSI-11. Keys for tagging and detagging the record tags must be passed from the access control center to enable the guard to verify tags.

There are five processing operations.

(1) The tables must be set up for tagging and detagging. The use of these tables will be described subsequently in this chapter.
(2) The initial flags must be cleared so as to allow external equipment to activate the interrupts.
(3) The keyboard inputs must be started; that is, the mechanism for monitoring the keyboard must be initialized.
(4) The mechanism for communication with the central processing unit of the PDP-11/45 must be set up.
(5) The Communications Pool (COMPOOL) initial conditions must be established.

5.2.4 Read Keyboard. Table 4 described the operation, purpose, inputs, processing, and outputs of the Read Keyboard segment. The name is descriptive of the operation. It reads the keyboard. The segment

Table 4. Read Keyboard

Operation:	Read'KBRD
Purpose:	Initialize keyboard read and waits until operator has completed an entry
Inputs:	• Flags from keyboard read routine
Processing:	• Transfers data to keyboard input buffer
Outputs:	• Keyboard input buffer

initializes the keyboard read and waits until the operator has completed. Concurrently, it records the message entered by the operator into the input buffers.

The inputs are from the operator via the keyboard and are comprised of flags from the keyboard read routine.

The processing consists of transforming the input characters from a serial stream into a sequential message stored in the input buffer. The outputs of the software segment are the keyboard inputs contained in the input buffer.

5.2.5 Process Entry. Table 5 shows the operation, purpose, inputs, processing, and outputs of the process entry software. Before the message is assembled in its final form, certain processing takes place in the LSI-11. The purpose is to provide initial scanning of operator entry, to decode key words and provide error messages for improper messages. The process entry routine also accepts inputs or changes to the data base in the form of modifying, appending, changing, or entering a new record into the data base.

The inputs to the routine are the messages assembled in the keyboard entry buffer.

The processing consists of six steps.

(1) The entry is scanned for key words.
(2) The key words are replaced with mathematical operators.
(3) The message from the keyboard is packed into a more dense form.
(4) The program checks for an illegal entry, that is, an operator error in entering the command. If an error is detected, it is indicated to the operator and to the central processing unit, 11/45.
(5) Control is returned to the exit guard.
(6) The original operator inputs and changes made by the program are placed in a separate buffer.

Table 5. Process Entry

Operation:	Process Entry
Purpose:	Provide initial scanning of operator entry. Decode key words and provide error messages if not a proper entry. Accept inputs or changes to data base.
Inputs:	• Keyboard entry buffer
Processing:	• Scan entry for key words • Replace key words with operators • Pack keyboard entry • Check for a legal entry as defined in 9340-77-74 • If an error is detected indicate to the operator and 11/45. Then return control to exit'guard • Channel inputs and changes into a separate buffer
Outputs:	• Scanned keyboard entry buffer • Error message to operator • Error message to access control center

The output of the software is the processed message placed in the scan keyboard entry buffer. Detected messages are output to the operator. Error messages are also communicated to the access control center and monitored to determine if the operator is deliberately trying to deceive the system.

5.2.6 Output to the 11/45. Once the operator message has been accepted by the exit guard, it is passed by the guard to the central processing unit, the 11/45. Table 6 shows the operation, purpose, inputs, processing, and outputs of the software segment.

Table 6. 11/45 Output

Operation:	Output'to'45
Purpose:	• Tag scanned keyboard entry buffer
Inputs:	• Scanned keyboard entry buffer • Key tables
Processing:	• Tag record • Request output to 11/45
Outputs:	Tagged record(s) to 11/45

The purpose of this program is to tag the scan keyboard entry buffer when such entries are comprised of new records to be entered into the data base. The initial tagging is necessary so that all records in the data base have a tab to enable retrieval. Once the records have been tagged, they are placed in a buffer for output to the 11/45.

The inputs to the program are the scanned keyboard entry buffer (output of the preceding program) and the key tables necessary to create tags for new record entries.

Processing is comprised of tagging records which will be described in detail subsequently, and a request for output to the 11/45. Outputs from the program include tag records to the 11/45.

5.2.7 Input Results. Table 7 shows the operation, purpose, inputs, processing, and outputs of the input results program.

Once the query has been communicated to the 11/45, the LSI-11 awaits input results. Inputs from the 11/45 are preceded by a signal. A transfer is initiated to the 11/45 input buffer of the messages which responded to the query.

The processing consists of waiting for the indication of results and the actual transfer of data from the 11/45. The outputs consist of messages contained in the 11/45 input buffer.

Table 7. Input Results

Operation:	Input'Results
Purpose:	• Wait for results from 11/45
Inputs:	• Input signal from 11/45
	• 11/45 input buffer
Processing:	• Wait for results indication
	• Transfer data from 11/45
Outputs:	11/45 input buffer

5.2.8 Verify Tag. Table 8 shows the operation, purpose, inputs, processing, and outputs from this program. It is assumed that the response to the input query resulted in a record being transferred from the 11/45. The purpose of the program is to operate the detag algorithm to establish whether or not the user will be granted access to the record which has been communicated.

Table 8. Verify Tag

Operation:	Verify'Tag
Purpose:	• Operate detag algorithm
Inputs:	• 11/45 input buffer
	• Key tables
Processing:	• Perform detagging algorithm
	• If a good tag operate good'tag
	• If a bad tag operate bad'tag
Outputs:	• Results of detagging algorithm

The inputs are the record from the 11/45 and the key tables necessary in the detagging algorithm, to be described subsequently. The processing is the execution of the detag algorithm. If the tag is good, the program branches to the good tag branch. If it is bad, the program branches to the bad tag branch. The outputs consist of results of the detagging algorithm.

5.2.9 Good Tag. Table 9 shows the operation, purpose, inputs, processing, and outputs of the Good Tag Program. Its purpose is to output results to the operator. Results in this case are a record retrieved from the 11/45 in reponse to a query initiated by the operator. The inputs are the message communicated from the 11/45 to the input buffer and results of detagging.

Table 9. Good Tag

Operation:	Good'tag
Purpose:	Output results to operator
Inputs:	• 11/45 input buffer
	• Results of tagging
Processing:	• Initiate output to operator
	• Return control to exit'guard
Outputs:	• Results of retrieval to operator

Processing consists of communicating the output records of the operator and returning control to the exit guard. The net result is the communication of those records retrieved to the requesting operator.

5.2.10 Bad Tag. Table 10 shows the operation, purpose, inputs, processing, and outputs of the Bad Tag Program. Its purpose is to block unauthorized data from being presented to the operator; that is, if the operator requested information from an unauthorized compartment the detagging algorithm would detect this and branch to the bad tag program.

Table 10. Bad Tag

Operation:	Bad'tag
Purpose:	• Block unauthorized data from presented to the operator
Inputs:	• 11/45 input buffer
	• Detagging Results
Processing:	• Return control to exit'guard
Outputs:	Output to access control center, ACC

The inputs are the message communicated from the 11/45 to the input buffer and the results of the detagging algorithm.

Processing consists of returning control to the exit guard without communicating the record to the operator. The output from this program is a message to the access control center (ACC).

5.3 PDP-11/45 Top Level Flow. Control to the data base system is through the PDP-11/45. Messages such as QUERY THE DATA BASE, DELETE RECORD, CHANGE RECORDS, etc., are communicated by the LSI-11 and executed in the 11/45.

5.3.1 Top Level 11/45 Flow. Figure 19 shows the flow diagram at the top level of the 11/45. The flow is divided into two segments: initialization and operation. Messages from the LSI-11 guard are processed by the input message lock. Next, the source of the message must be identified. There are five possible sources: Console 1, which is an LSI-11 No. 1; Console 2, which is an LSI-11 No. 2; Console 3, which is an LA-36 teleprinter; the card reader, or an unknown source. Once the message source has been identified, the actual program will run in the partition of that source so as to achieve multiprogramming.

5.3.2 Initialize 11/45. Table 11 shows the operation, purpose, inputs, processing, and outputs of the Initialize program. The purpose, as

Table 11. Initialize 11/45

Operation:	Initialize '45
Purpose:	• Initialize 11/45 and LSI-11's
	• Initialize all I/O drivers
	• Use card reader to set up keys tagging
	• Check disk protection codes
Inputs:	• Initial conditions file
	• Keys from card reader
Processing:	• Set up I/O drivers
	• Set up tagging I/O tagging keys
	• Start access control center (ACC) module
	• Start 11/45 and LSI systems
Outputs:	• Key tables
	• Initial conditions

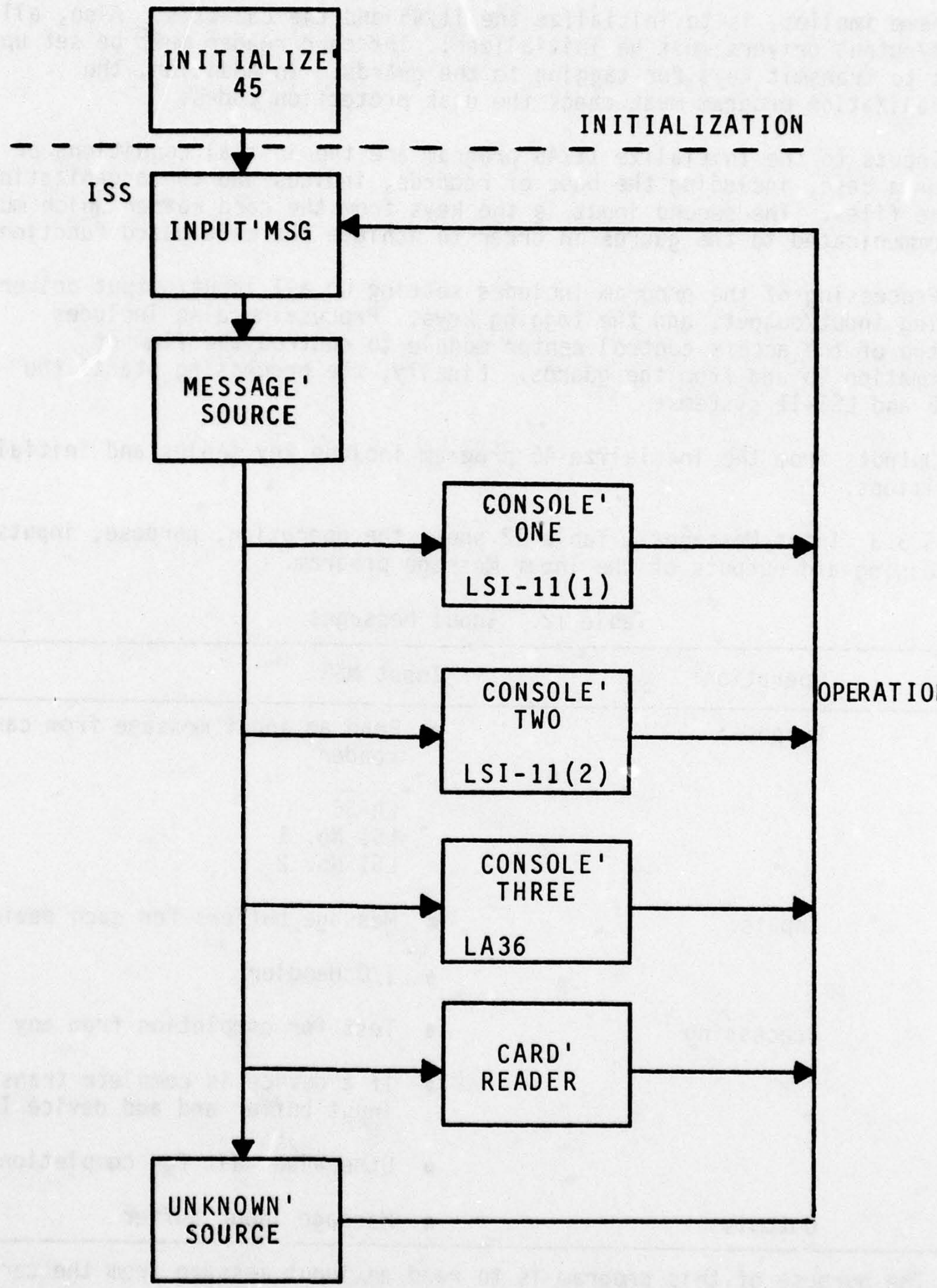


Figure 19 PDP-11/45 ISS TOP LEVEL FLOW

the name implies, is to initialize the 11/45 and the LSI-11's. Also, all input/output drivers must be initialized. The card reader must be set up so as to transmit keys for tagging to the guards. In addition, the initialization program must check the disk protection codes.

Inputs to the Initialize 11/45 program are the initial conditions of the data base, including the base of records, indices and the organization of the files. The second input is the keys from the card reader which must be communicated to the guards in order to achieve the exit guard function.

Processing of the program includes setting up all input/output drivers, tagging input/output, and the tagging keys. Processing also includes startup of the access control center module to control the flow of information to and from the guards. Finally, the processing starts the 11/45 and LSI-11 systems.

Outputs from the initialize 45 program include key tables and initial conditions.

5.3.3 Input Messages. Table 12 shows the operation, purpose, inputs, processing and outputs of the Input Message program.

Table 12. Input Messages

Operation:	Input'MSG
Purpose:	• Read an input message from card reader LA-36 LSI No. 1 LSI No. 2
Inputs:	• Message buffers for each device • I/O Handler
Processing	• Test for completion from any • If a device is complete transfer input buffer and add device ID • Otherwise wait for completion
Outputs:	• Message inpout buffer

The purpose of this program is to read an input message from the card reader, the LA-36, or either of the two LSI-11's. Inputs to the program include message buffers for each driver and an input/output handler.

Processing tests for the completion of any device. If a device is completed, the message is transferred to the input buffer and the device identification is added. If no devices are complete, the program waits for completion. Outputs from the program include messages in the input buffer.

5.3.4 Message Source. Table 13 shows the principal components of the message source program.

Table 13. Message Source

Operation:	Message' source
Purpose:	Test message source and transfer control to proper operation
Inputs:	• Message input buffer
Processing:	• Test message source and give to one of the following: - Console'one - Console'two - Console'three - Card'reader
Outputs:	• None

The purpose of the software is to test the message source and transfer control to the proper operation; that is, the partition to be run on the multiprogramming.

Inputs to the program include messages residing in the input message buffer. Processing is comprised of testing message source and giving control to one of the following: Console No. 1, Console No. 2, Console No. 3, or Card reader. There are no outputs from this program.

5.3.5 Console One. The Console One program is intended to process all messages originating from that console. The purpose is to process Console No. 1's query; that is process in the sense of querying the data base or executing other commands originating at that console.

Processing is comprised of executing Console No. 1's query by tasking the query processing programs. The query processing programs are large and complex and are not covered in this section. After the query has been processed, control is returned to the main program. Outputs include the proper responses to the operator's query. Table 14 shows the major components of the Console One program.

Table 14. Console' One

Operation:	Console'one
Purpose:	Process console one's query
Inputs:	Console'one message buffer
Processing:	• Process console one's query by tasking query'processing • Return control to ISS
Outputs:	Query response

5.3.6 Console Two, Three, Card Reader. Table 15 highlights the functions of the other three programs that process sources of operator messages into the PDP-11/45.

Table 15. Console Two

Operation:	Console'two Console'three Card'reader
Purpose:	Process messages originating at consoles
Inputs	Console'two, console'three message buffer
Processing:	• Process console two and console three's query by tasking query' processing • Return control to ISS
Outputs:	Query response

The sources are the other consoles and the card reader. The inputs, processing, and outputs are identical to that of Console No. 1, except for servicing a different terminal.

5.3.7 Unknown Source. Table 16 shows the operation, purpose, inputs, processing, and outputs of the Unknown Source Program. This software is included to complete the processing of queries, even though identification of the message origin cannot be established. The reason for including this program is to provide an input to the access control center to detect attempts by malicious users to penetrate the data base system.

Table 16. Unknown Source

Operation:	Unknown Source
Purpose:	• Process a message from an source
Inputs:	Message input buffer
Processing:	• Count unknown messages • If count exceeds a specific log that a penetration attempt been made • Return control to ISS
Outputs:	• Unknown message counts • Penetration message

The inputs are from the message buffer. Processing consists of counting unknown messages. If the count exceeds a specific number, log a penetration attempt and return control to the central progam. Outputs to the access control center include unknown message counts and penetration attempt warnings.

5.4 Query Processing. A major portion of software is devoted to the execution of data base transactions in the PDP-11/45. This processing is called query processing and is described in this section.

5.4.1 Top Level Flow. Figure 20 is a flow diagram of the overall query processing. The six blocks show the sequence of processing. Some of the processing is very complex and will be described in more detail in the appendix.

The processing operates on messages from one of the known sources. First the request is accepted, then it is passed to the generate intermediate language program. This is a reverse Polish notation scan that breaks the command up into a processing sequence.

If there are errors in the command, administrative error messages are sent back to the user, and the program exits. If there are no errors, the intermediate code is executed and the transaction results, such as updating or querying the data base. When processing results and messages to be communicated to the operator such as records matching the query key and conditions, the records are sent back to the user, and the program exits.

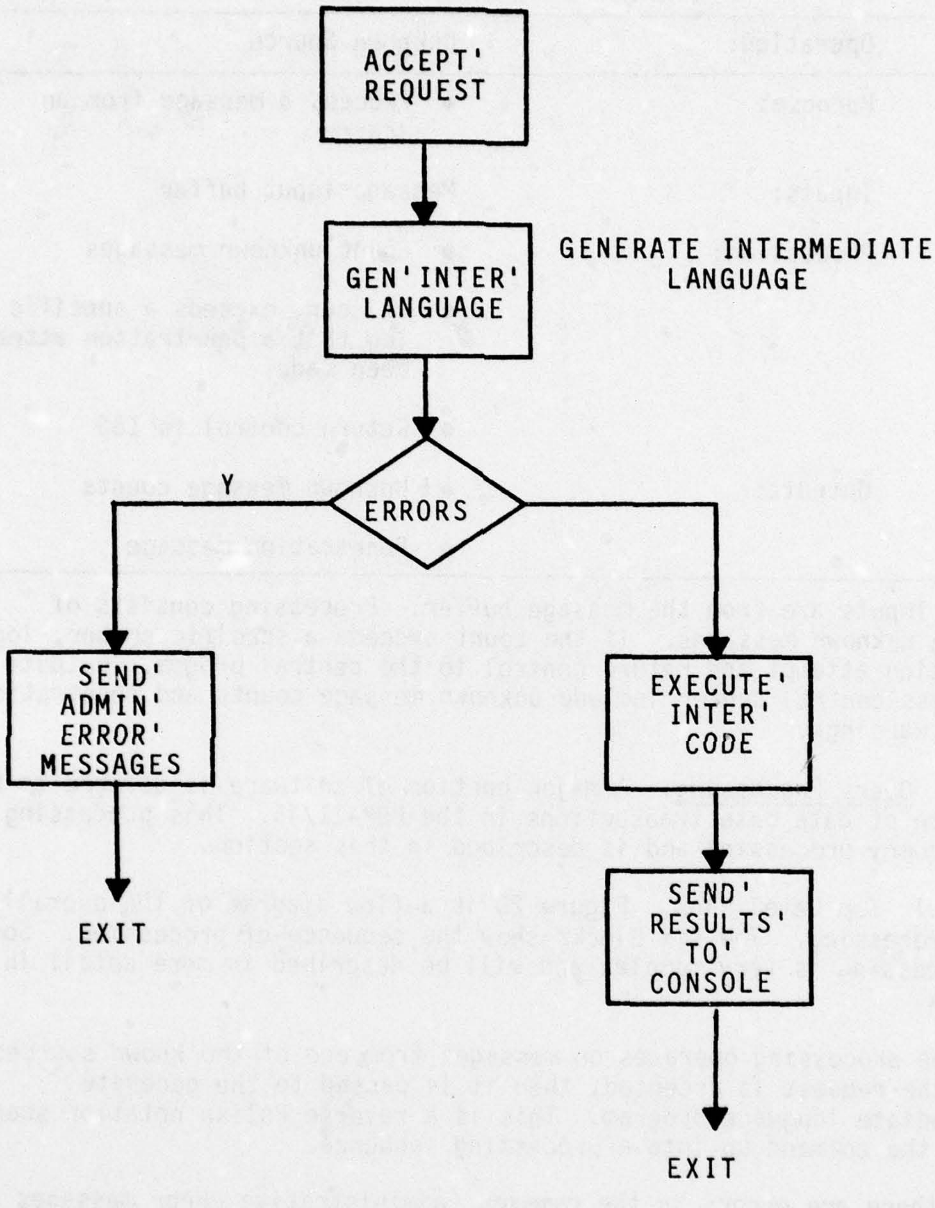


Figure 20 QUERY PROCESSING

5.4.2 Accept Request. Table 17 outlines the performance of the Accept Request program. Its purpose is to set up message buffer pointers for the generate intermediate language. Messages come in to the program via the message buffer, and the processing sets up pointers to the message buffer. Outputs are pointers for the generate intermediate language.

Table 17. Accept Request Number

Operation:	Accept'request
Purpose:	• Set up message buffer pointers for gen'inter'language
Inputs:	• Message buffer
Processing:	• Set up pointers to message buffer
Outputs	• Pointers for gen'inter'language

5.4.3 Generate Intermediate Language. Table 18 shows the operation, purpose, inputs, processing, and outputs of the Generate Intermediate Language program. The purpose is to scan query requests and generate intermediate code for execution by the intermediate language. Inputs to the program include message buffer pointers and communications pool data definitions.

Table 18. Generate Intermediate Language

Operation:	Gen'inter'language
Purpose:	• Scan query request and generate intermediate code for execution by execute'inter'language
Inputs	• Message buffer pointers • Compool data definitions
Processing:	• Perform a reverse Polish notation scan on the query and generate operational intermediate code • If there are errors, do the following: - Count errors - Make a list of error numbers
Outputs:	• Intermediate code • Error count • Error numbers

Processing on the command includes performing a reverse Polish notation scan of the query and generating an operational intermediate code. If there are errors in the command, the program counts the errors and makes a list of error numbers. Error messages will be communicated back to the originating operator. Outputs of the program include intermediate code, error count, and error numbers.

5.4.4 Errors. Table 19 shows the operation, purpose, inputs, processing, and outputs of the Errors program. Its purpose is to make a decision whether to operate the intermediate code. This is done only if there are no errors. Inputs to the program include intermediate code, error count, and error numbers from the preceding program. Processing proceeds if the error count is zero. In this case, control is given to the execute intermediate code program. If the error is greater than zero, control is passed to the program entitled, "Send Administrative Error Messages." There are no outputs from this program as it simply tests for errors.

Table 19. Errors

Operation:	Errors
Purpose:	• Make decision on whether to operate intermediate code
Inputs:	• Intermediate code • Error count • Error numbers
Processing:	• If error count is zero, give control to execute'inter'code • If error count is greater than zero, give control to send'admin 'error'messages
Outputs:	None

5.4.5 Send Administrative Error Messages. Table 20 shows the principal performance features of the Send Administrative Error Messages program. As the name implies, its purpose is to communicate errors back to the originator, when detected by the command interpreter. Input to the program includes error count, error numbers, message buffer, and error message file.

Processing consists of reading an appropriate message from the error message file and sending an error count to the console for each error detected. The program updates the error count for the console that

AD-A054 508 HARRIS CORP MELBOURNE FLA ELECTRONIC SYSTEMS DIV F/G 9/2
INTELLIGENCE SECURITY SUBSYSTEM.(U)
MAR 78 F ANDERS, W MALL, R MCGILL F30602-76-C-0445
UNCLASSIFIED RADC-TR-78-33 NL

2 OF 4
AD
A054508

originated the query. If the error count exceeds threshold, a warning message is sent to the access control center to provide an early indication of a malicious user operating at the terminal. Outputs from the program are error messages to the operator, error count for console, and access control center warning.

Table 20. Administrative Error Messages

Operation:	Send'admin'error'messages
Purpose:	Send operator error messages
Inputs:	• Error Count • Error numbers • Message buffer • Error message file
Processing:	• For each error read appropriate message from error message file and send to console • Update error count for console • If error count exceeds threshold, send ACC warning
Outputs:	Error messages to operator Error count for console ACC warning

5.4.6 Execute Intermediate Code. Table 21 shows the operation, purpose, inputs, processing, and outputs of the Execute Intermediate Code program. The program is a large and complex interpretation of user messages into the data base. Its purpose is to execute the query request. Inputs to the program include intermediate language code.

Processing consists of executing intermediate code, storing the hit count, and the record number of the originating console in the hit table. Outputs from the program include hit count; that is, the number of records that meet the search criteria, and a hit table which lists the data base address next to each record.

5.4.7 Send Results to Console. Table 22 lists the major functions of the Send Results to Console Program. Inputs to the program include the hit count and hit table.

Table 21. Execute Intermediate Code

Operation:	Execute 'inter'code
Purpose:	Execute query request
Inputs:	Intermediate code
Processing:	• Execute intermediate code • Store record number in hit table for appropriate console • Store hit count
Outputs:	• Hit count • Hit table

Table 22. Results to Console

Operation:	Send'results to'console
Purpose:	Send results of query to console
Inputs:	Hit count Hit table
Processing:	• If hit count greater than zero, send all records in hit table to console • Otherwise send no response admin message to console • Exit
Outputs:	• Messages to console

Processing executes the command originated by the console operator. Specifically, if the hit count is greater than zero, send a'l records in the hit table to the console. Otherwise, send no administrative messages to the console. Outputs from the program include messages to the console in the form of records that meet the search criteria.

5.5 <u>Data Base Query Operations</u>. Thus far the flow diagrams have presupposed a language that the operator uses to interrogate and modify the data base. That language will now be described in this section.

5.5.1 Storage and Retrieval Block Diagram. Figure 21 shows the storage and retrieval block diagram. The data base is contained on disk file and is stimulated by query inputs. The queries result in a search to find the referenced records with the results of the query communicated to the operator. The data base may be updated either by the operators or by batch transactions.

In the enciphered tag approach, the record is broken up as it appears on the right-hand side of Figure 21. The first word of the record message packet contains the length of the record. The second word of the record contains the length of the data. The third word contains a description of the data; that is, whether it is an event or report. This is followed by the information itself and is used by the data base guard to control access.

5.5.2 Terminal Input/Output Block Diagram. The terminal illustrated in Figure 22 is the means of communication between the operator and the feasibility model. The operator inputs messages, and the terminal outputs the messages, either by printing or displaying them. The terminal includes the data base guard, which is responsible for the data tagging and detagging algorithms. The keyboard is the means of man/machine communication and the display presents the output results.

5.5.3 List of Statements. The language by which the operator communicates with the data base system allows a great deal of flexibility and is a powerful tool for querying and updating the data base. There are five statements:

- Display
- Print
- Delete
- Add
- Change

5.5.4 The Display Statement. The Display statement allows the operator to query the data base and select records which match the correlation criteria. Display has the following format:

Display	All Type	If Condition

where:

Find	=	Find statement
Type	=	Specific record type (such as report)
All	=	All records which qualify
If	=	If statement
Condition	=	If statement condition

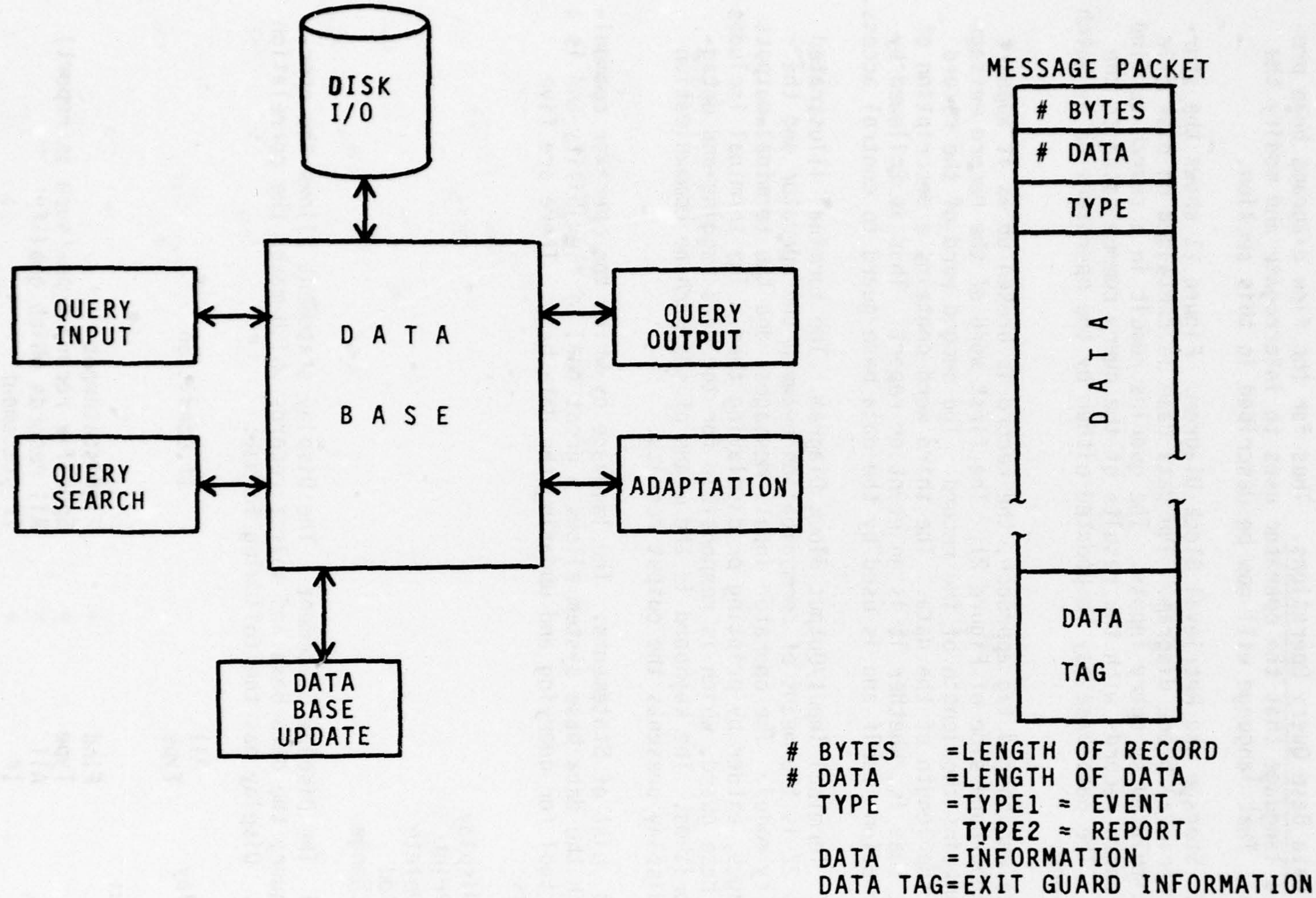


Figure 21 STORAGE AND RETRIEVAL BLOCK DIAGRAM

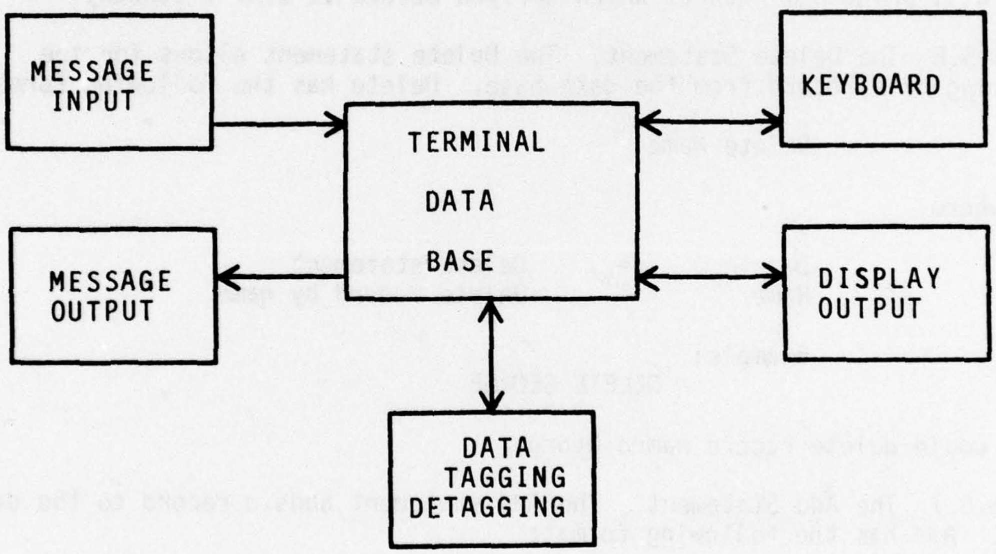


Figure 22 TERMINAL I/O BLOCK DIAGRAM

Example:

DISPLAY REPORT IF NAME EQ GEORGE

This will search the data base for a report with the name of George and display it if found.

5.5.5 The Print Statement. The Print statement allows the operator to print instead of display a record. Print has the same format as the display statement, except print replaces display.

Example:

PRINT ALL IF TIME LS 1 1200

This will print all records which arrived before 12 a.m. 1 January.

5.5.6 The Delete Statement. The Delete statement allows for the deleting of a record from the data base. Delete has the following format:

Delete Name

where

Delete	=	Delete statement
Name	=	Delete record by name

Example:

DELETE GEORGE

This would delete record named George.

5.5.7 The Add Statement. The Add statement adds a record to the data base. Add has the following format:

Add	Report Event	Name = nnnnn data =

where

Add	=	Add statement
Report	=	Report type record
Event	=	Event type record
Name	=	Name identifier
nnnnn	=	Name of record (up to 30 characters)
Data	=	Data identifier If report alphanumeric characters If event then parameters on Compool names of items in an event record.

Example:

ADD REPORT NAME = GEORGE DATA = This is a report about George. Date 1 January 1977.

This would add a report to the data base with the name "George" and the data as indicated above.

5.5.8 The Change Statement. The Change statement allows the operator to change a record. The Change statement has the following format:

Change Report/Event Name = nnnnn Data =

where:

Change	=	Change statement
Report	=	Change report
Event	=	Change event
Name	=	Name identifier
nnnnn	=	Name of record to be changed
Data	=	Data delemeter followed by Compool items or report text to be changed.

5.5.9. The If Statement. The If statement is used in conjunction with the display or print statement. If is the search argument of the Display and Print these statements. The format of If is:

If Condition

where:

If = If statement

Condition = Logical condition i.e., where:

CC	=	logical operator
EQ	=	equal
LS	=	less
GR	=	greater
LQ	=	less or equal
GP	=	greater or equal
CN	=	connectors

or

and

A, B, C, D ... N = are expressions

Example:

IF TIME GR 2140Z AND RANGE LS DISTANCE - 256

Condition is true if time is after 2140Z and range is less than distance -256 miles

5.6 Simplified Record Tagging and Detagging. Although the emphasis in this section is to describe one of the two methods, namely, the enciphered

tag approach, the other approach, enciphered record, will also be explored in the feasibility model.

The essential feature of the enciphered tag approach is that at the end of the record a tag appears which identifies the compartment from which the record was drawn. This compartment when deciphered can be compared with the access privileges of the user to determine whether the requested record should be passed on to the user. The principle of design is such that the tag cannot be forged in any way by a malicious user. It is assumed that the espionage agent cannot encrypt the record without the key. The simple encryption algorithm here is intended to illustrate the scrambling/descrambling program for purposes of studying timing and sizing of the guard. Subsequently, more advanced encryption algorithms will be used that will deny access for any period of decades, or even centuries.

The following points result from the assumption that the espionage agent cannot encipher the record when denied access to the key:

- Since the checksum is computed on the enciphered record, the espionage agent cannot compute the checksum. The agent can guess at it, but chances of guessing right are only 1 in 2^{32}.
- If the agent falsifies either the checksum or the record, such falsification will be detected by the guard when the calculated checksum is compared with that contained in the tag.
- Since the compartment identification is enciphered by combining it with the checksum, the agent cannot find out what the compartment is, and, further, cannot forge the compartment because it is both enciphered and has a checksum.

5.6.1 Key Initialization. The key is passed from the access control center to the exit guard. It consists of a string of 52 characters, as follows:

- Characters 1 through 40 equal all letters (no duplicates), all numbers (no duplicates), space, and special characters (any three).
- Characters 41 through 46 equal six-digit number which is a key to start a random number generator.
- Characters 47 through 52 equal 16 ones and zeros. A one indicates data access for this compartment.

5.6.2 Tagging Process. The tagging process consists of five steps, as follows:

1. Generate an 8-bit random number, RN.
2. Store RN in the tag.

3. Using temporary storage, encrypt the record as follows: for each character (8 bits), search the translation table for a match, If a match is found, replace the character with the index of the search. Increment RN. Logical exclusive OR the translated character with the random number table as indexed by RN.
4. Checksum the encrypted record and store in the tag.
5. Using the least significant 8 bits of the checksum as an index to the random number table, logically exclusive or the random number and store in the compartment.

5.6.3 Typical Record Tag. Figure 23 shows a diagram of a typical record. It is comprised of 16-bit words shown at the top of the diagram as Bit Positions 0 through 15.

There are N words in the record. The 0 position contains the number of words in the record, N. The last three words in the record contain the necessary information. The random number is contained in the low byte of word N -4. The compartment is stored in position N -3, and the checksum is contained in two words, Positions N -2 and N -1.

5.6.4 Detagging Process. The detagging process proceeds in six steps, as follows:

1. Fetch RN from the tag.
2. Encrypt record as described in Step 3 of the tagging process.
3. Computer checksum of the encrypted record.
4. Using the technique described by Step 5 of the tagging process, fetch the compartment.
5. Logical exclusive OR the key compartment and the compartment from 4 and store the result in the compartment.
6. If the checksums from the tag and result of 3 equal, and if compartment equals 0, the tag is correct.

5.7 Detailed Flow Diagrams - Query Processing. This section contains the detailed flow diagrams corresponding to the top level flow in Section 5.4 dealing with Query Processing. Each of the six blocks of Figure 20 will be discussed in more detail.

A subroutine named CCNTL acts in the capacity of supervisor for the Query Processor. A flow diagram of CCNTL is shown in Figure 24, as well as a correlation between it and the top level diagram Figure 20.

5.7.1 Accept Request. As stated in Section 5.4.2 the purpose of this block is to initialize the necessary tables and message buffer pointers for the generation of the reverse Polish code, which is the intermediate language. A detailed flow of the initialization function is given in Figure 25.

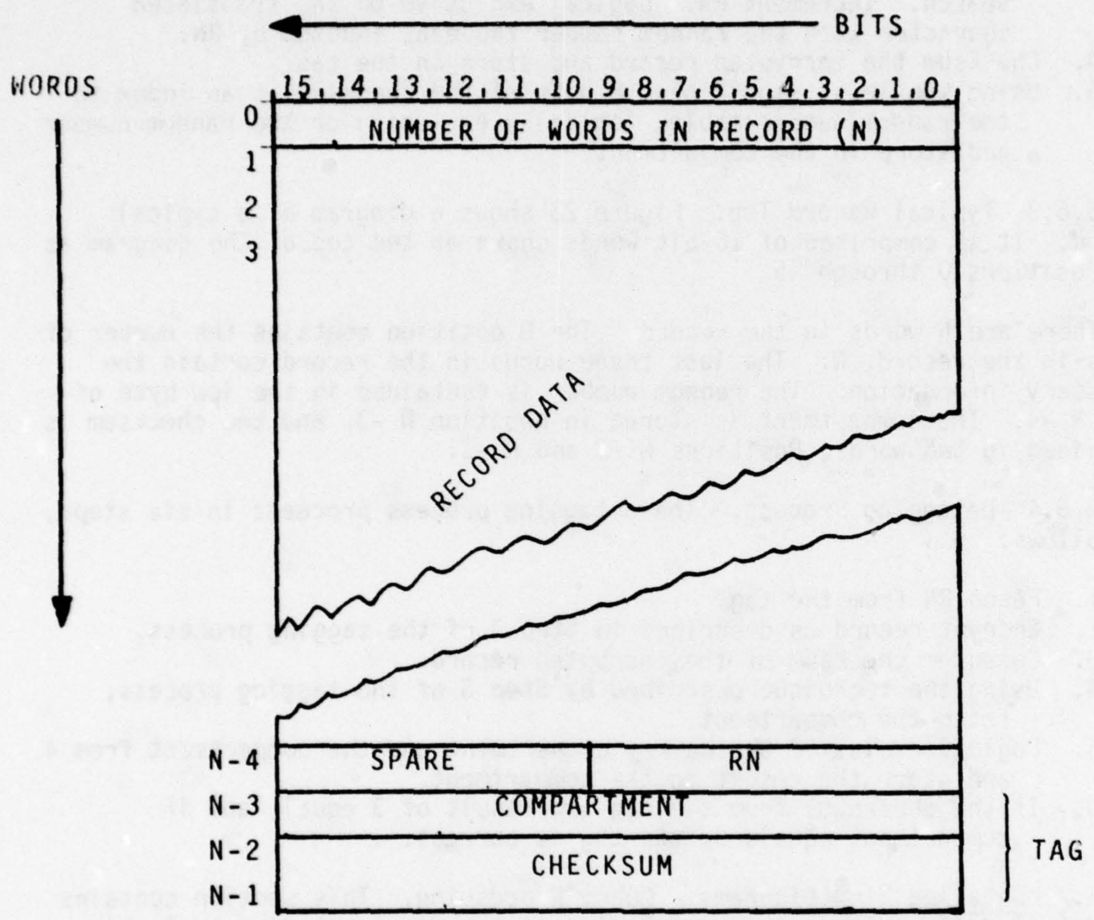


Figure 23

TYPICAL RECORD TAG

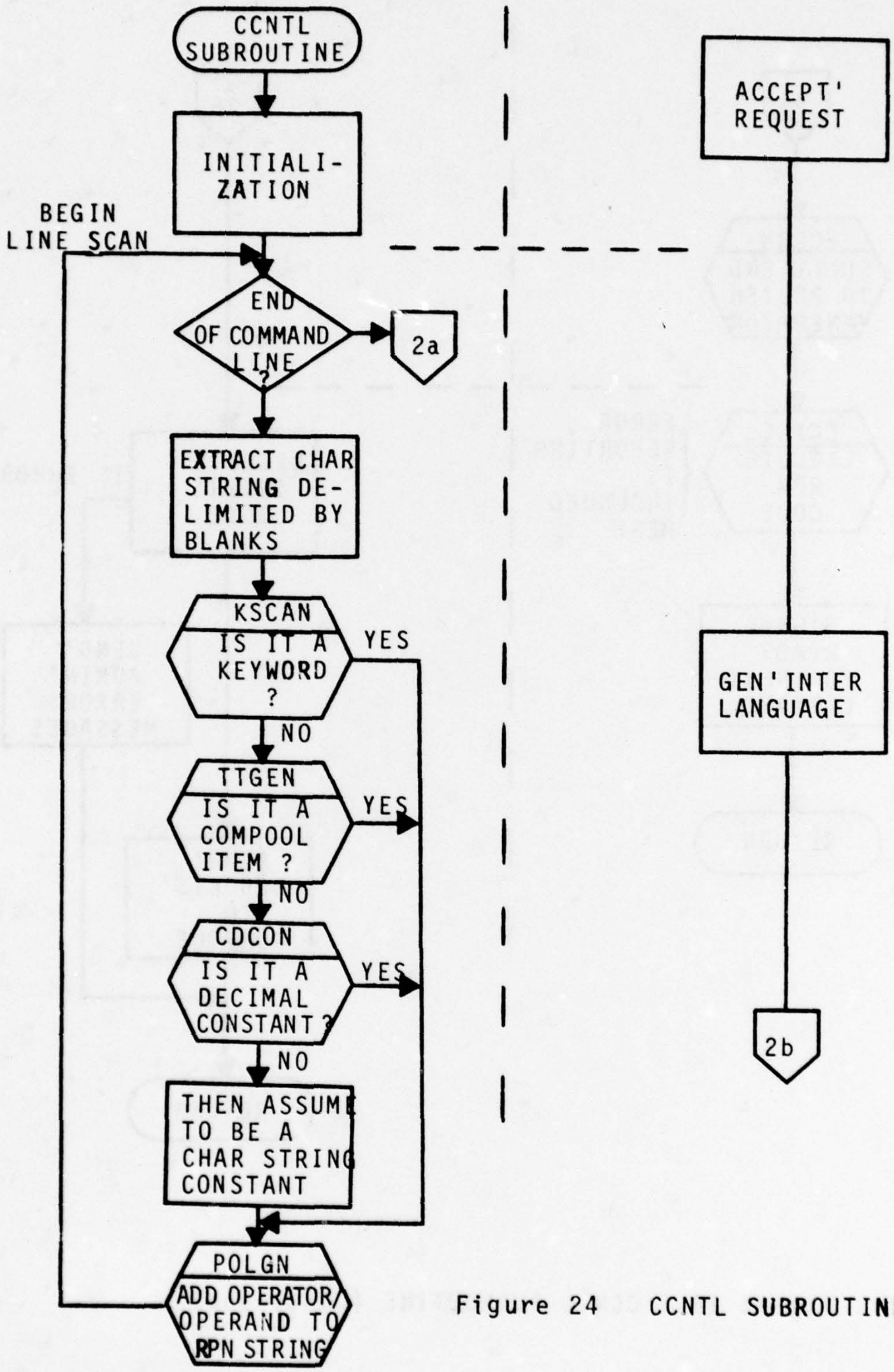


Figure 24 CCNTL SUBROUTINE

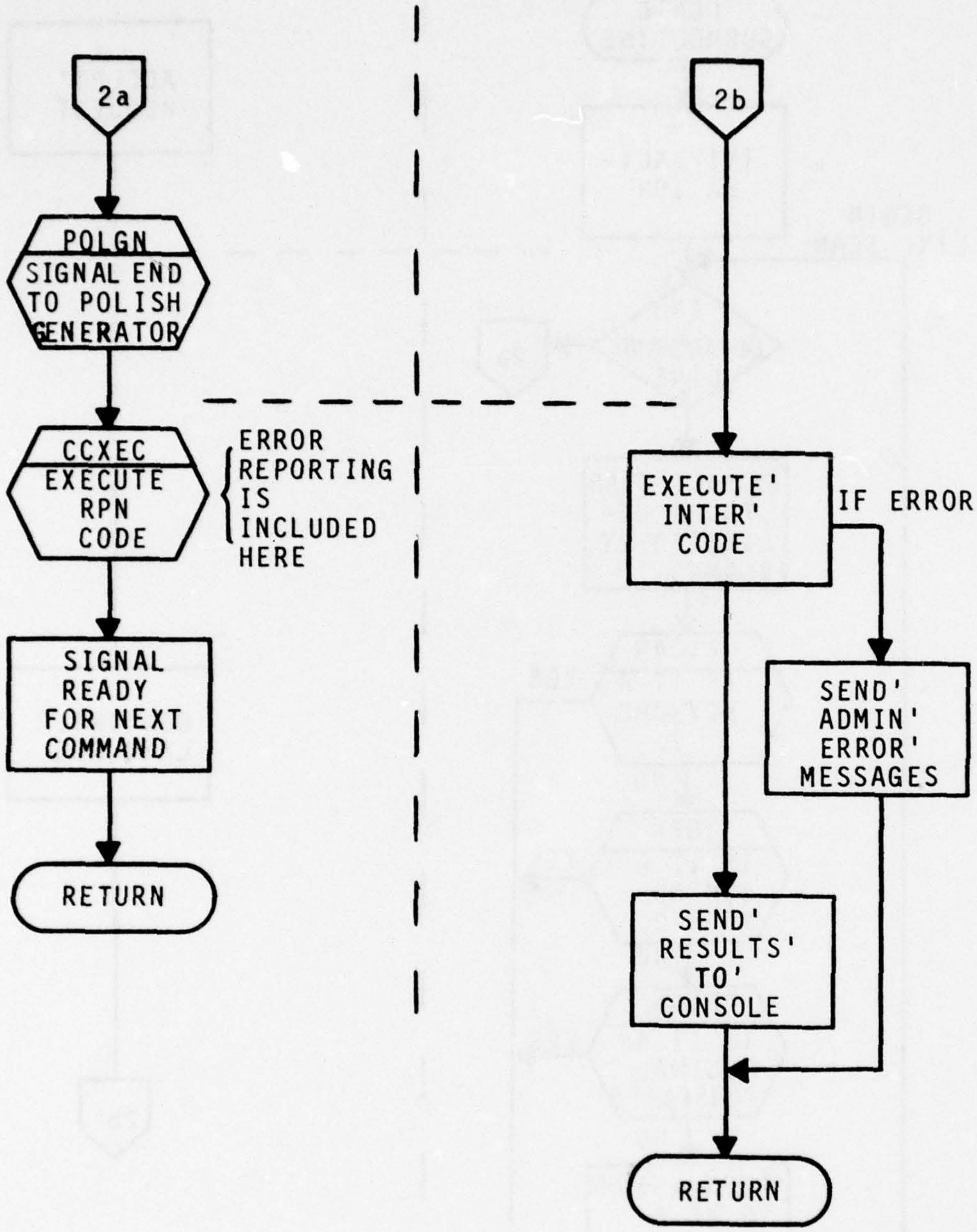


Figure 24 CCNTL SUBROUTINE (Continued)

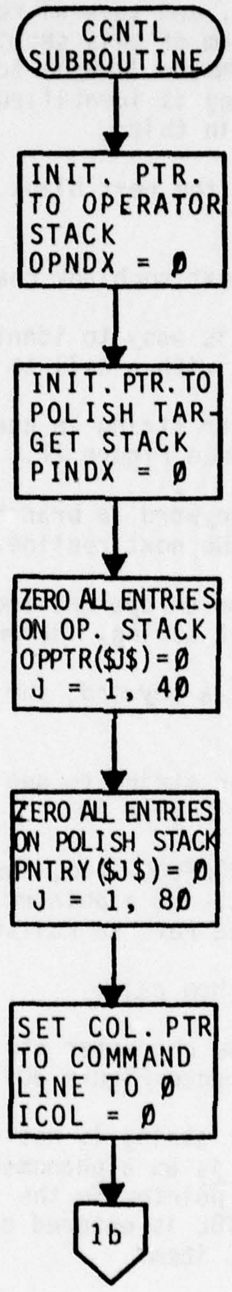


Figure 25 INITIALIZATION PHASE

5.7.2 Generate Intermediate Language. Generation of the reverse Polish code is a multistep process, and several routines are used to simplify the operation. A flow diagram of this section of the CCNTL can be found in Figure 26. First, the command line is scanned from left to right and each successive character string is identified and processed individually. Two Jovial operations aid in this:

BLNK - which searches for the next blank character in a command line.

NBLNK - searches for the next nonblank character in a command line.

With these two operations, it is easy to identify a character string and begin processing. This begins with a call to the following routine:

KSCAN - checks the character string in question to see if it is a keyword. Reference Figure 27.

If the character string is a keyword, a branch is taken to enter the keyword on the Polish string via the next routine.

POLGN - accepts an operator or operand and correctly places it into the Polish target string. Reference Figure 28.

If the character string is not a keyword, the following routine is called:

TTGEN - Compare a character string to see if it is a valid COMPOOL item. Reference Figure 29.

If the character string is a COMPOOL item, then the attributes of that item are stored in a table, TAGTBL, and a pointer to that entry in the table is saved and entered onto the reverse Polish string via POLGN.

If it is not a COMPOOL item, then call:

CDCON - Check to see if the character string is a decimal constant. Reference Figure 30.

In the case that the character string is not a decimal constant, it is assumed that the character string is an alphanumeric constant to be compared. The length and the column pointer to the string are stored in the TAGTBL and the pointer to the TAGTBL is entered onto the re erse Polish string via POLGN as with a COMPOOL item.

If the character string is a decimal constant, it is converted to binary and stored in the TAGTBL, with the pointer again to be entered on the reverse Polish string via POLGN.

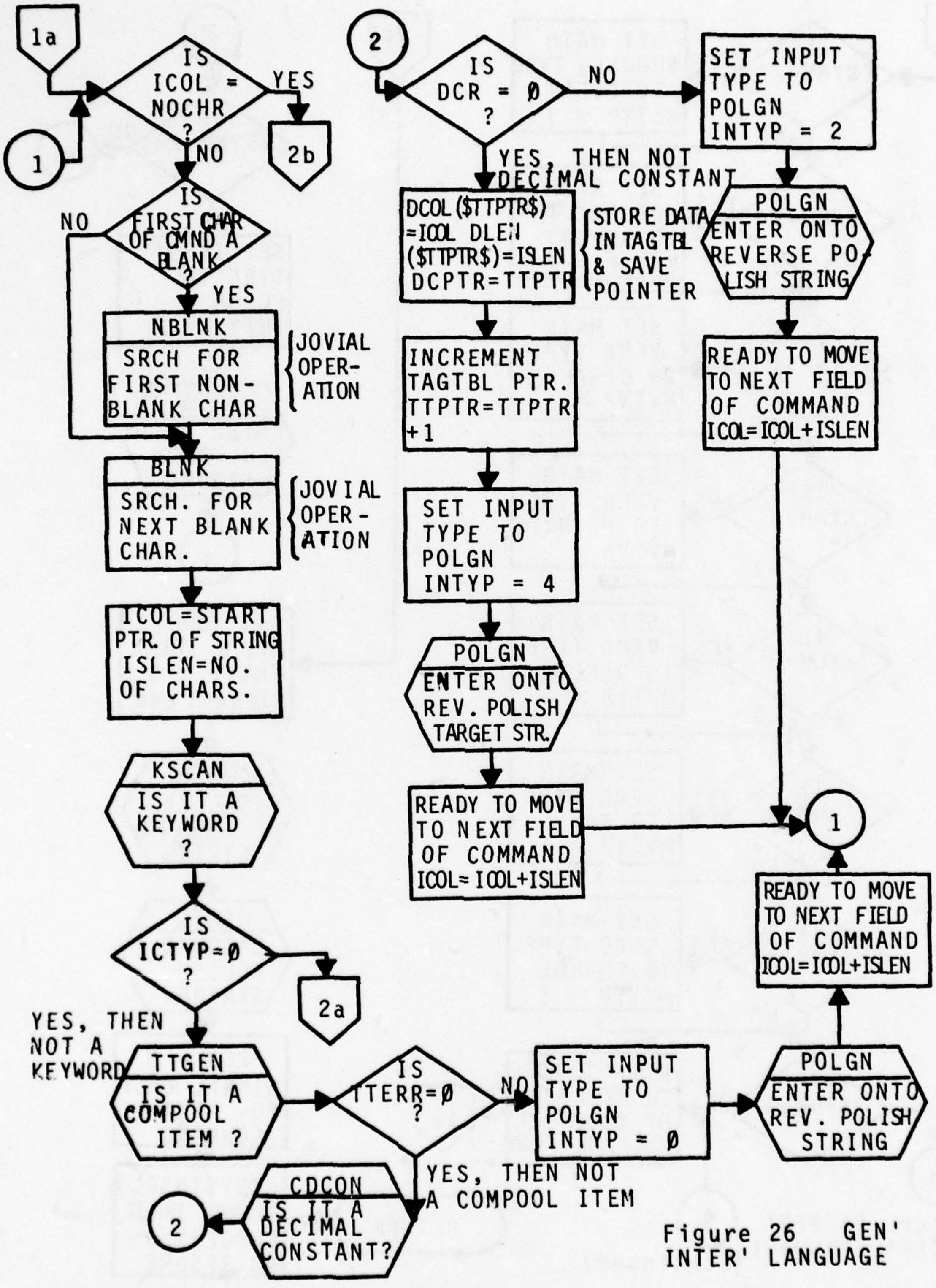


Figure 26 GEN' INTER' LANGUAGE

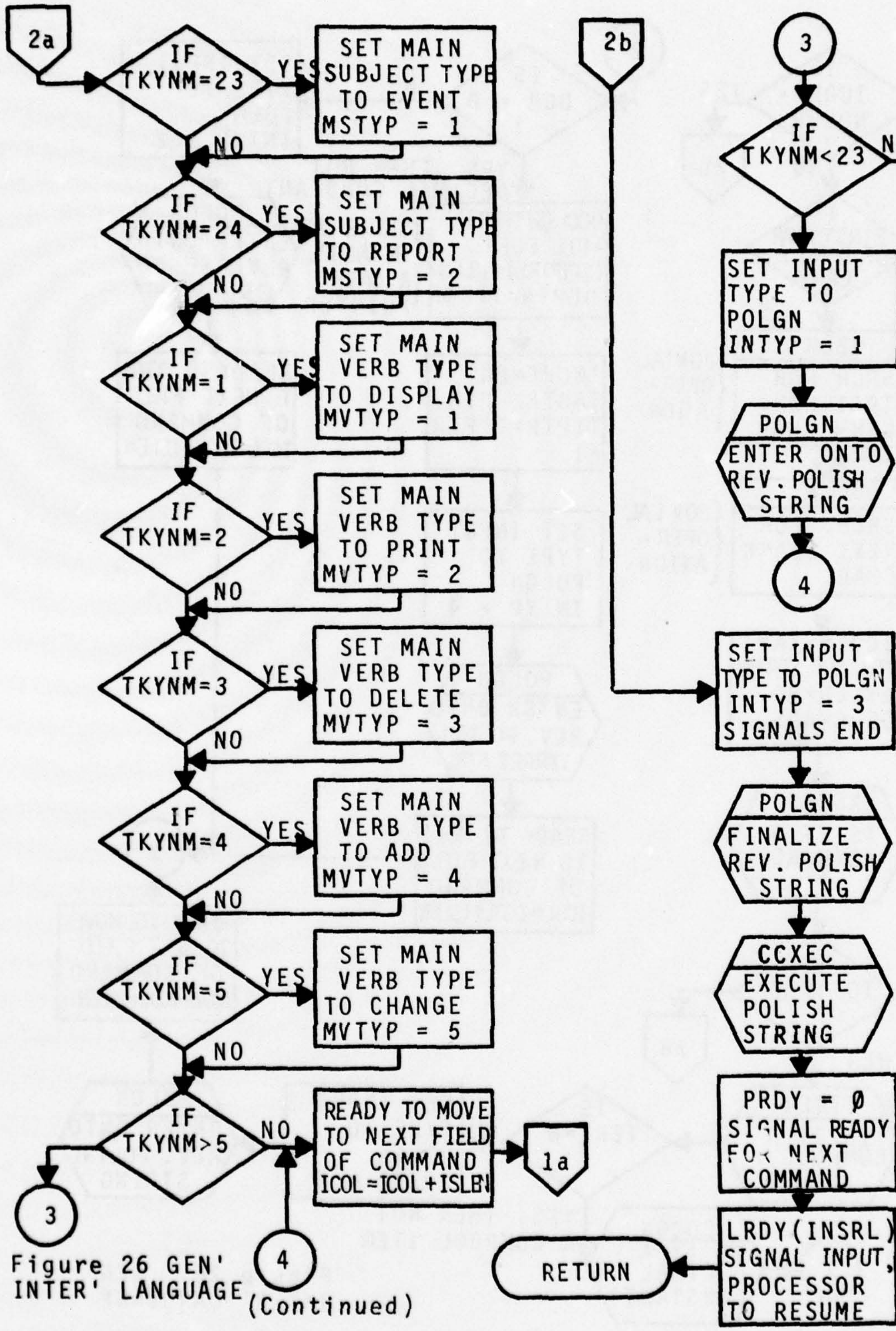


Figure 26 GEN' INTER' LANGUAGE (Continued)

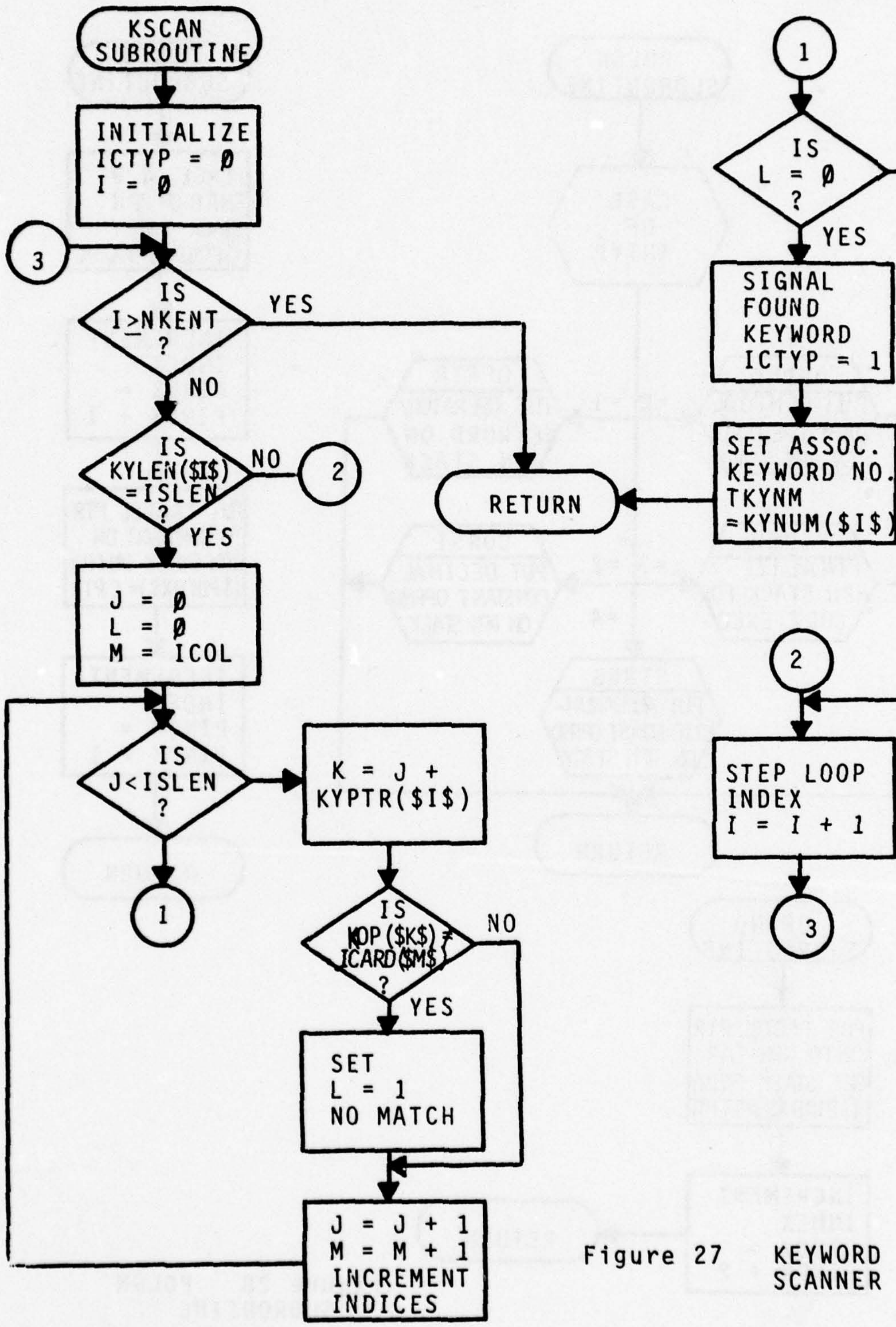


Figure 27 KEYWORD SCANNER

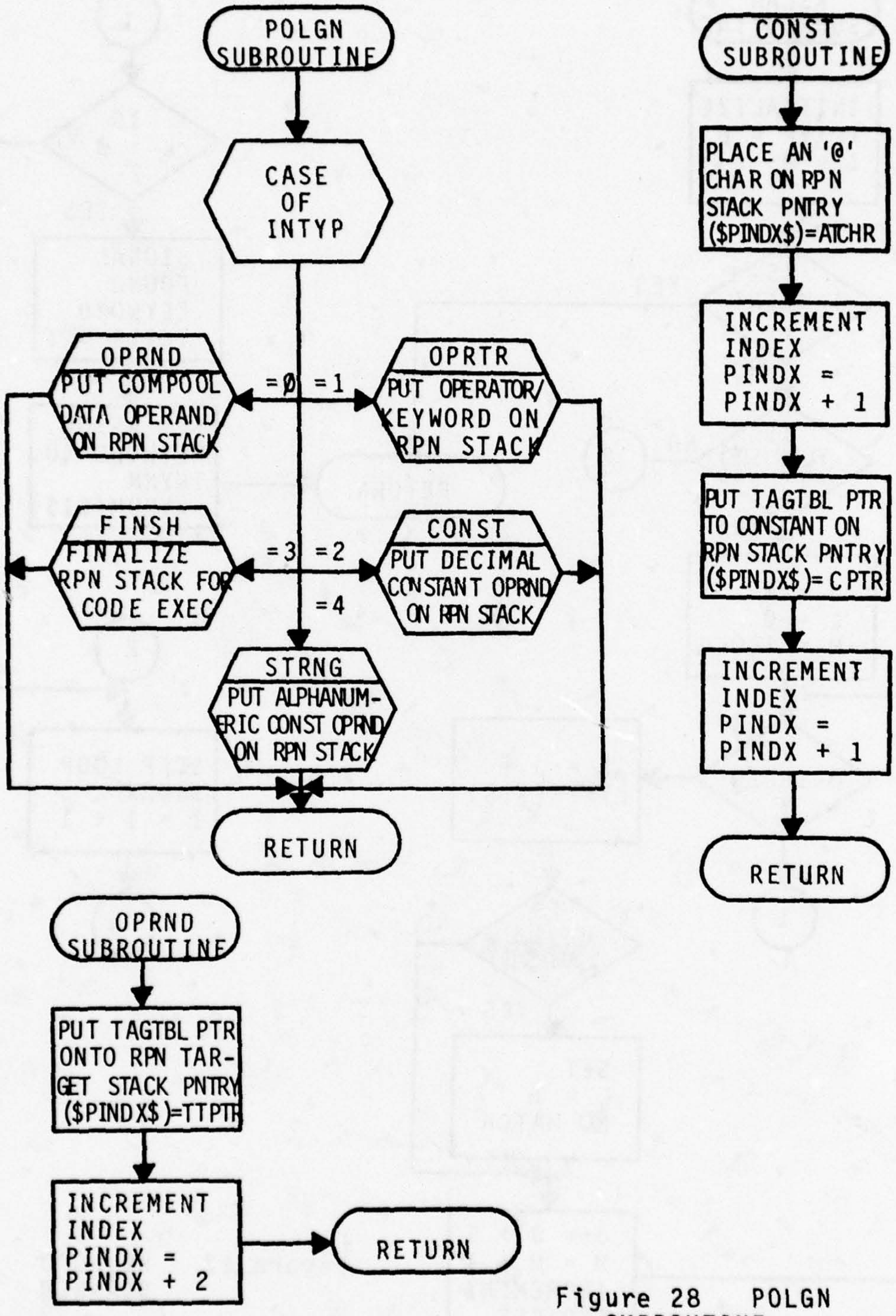


Figure 28 POLGN SUBROUTINE

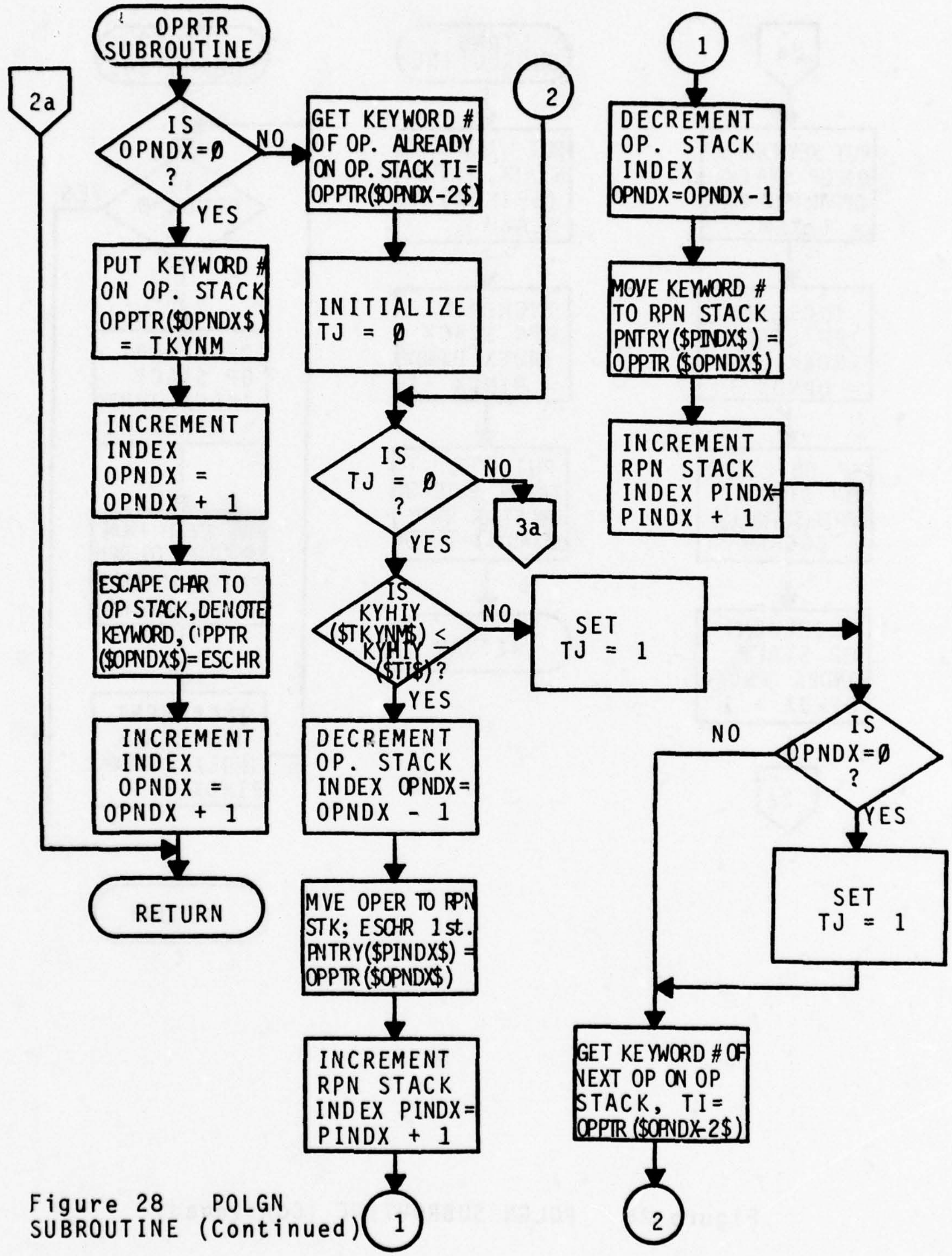


Figure 28 POLGN SUBROUTINE (Continued)

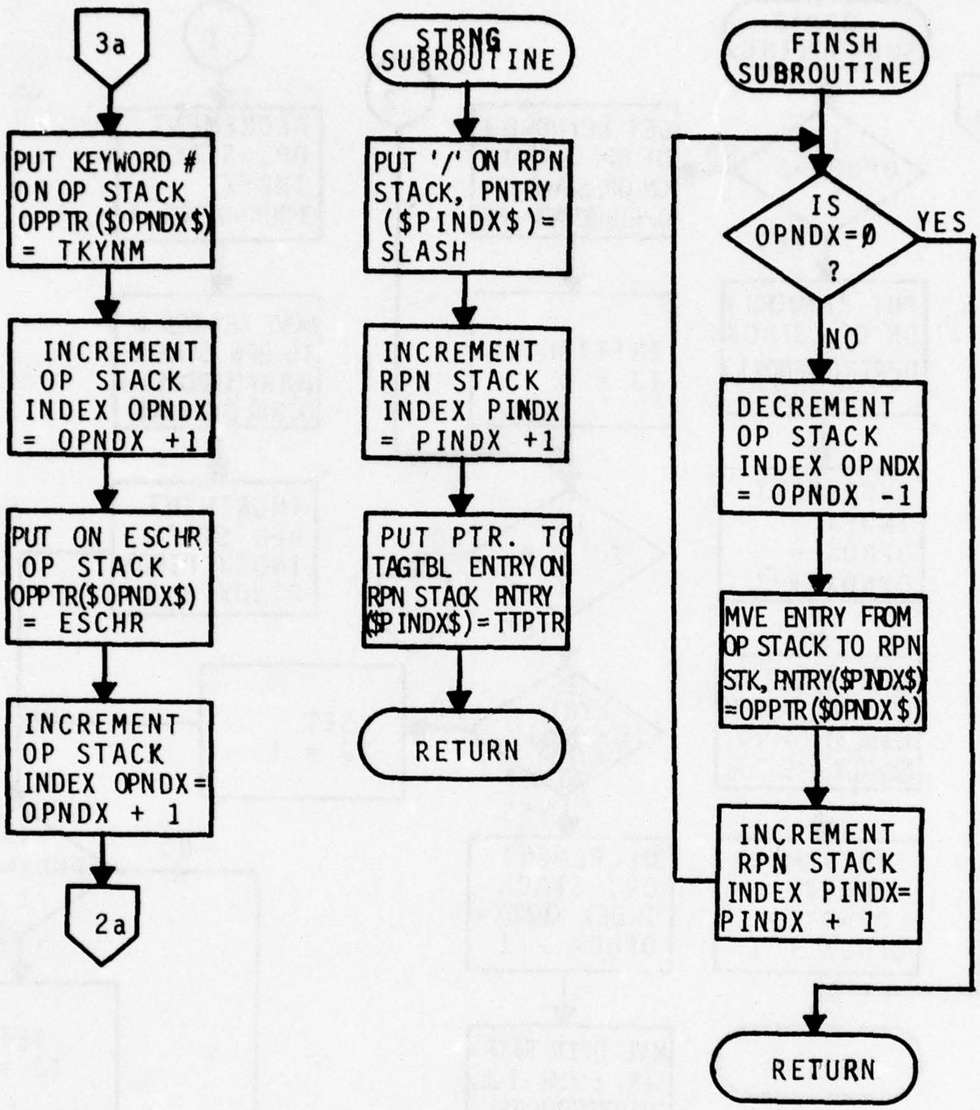


Figure 28 POLGN SUBROUTINE (Continued)

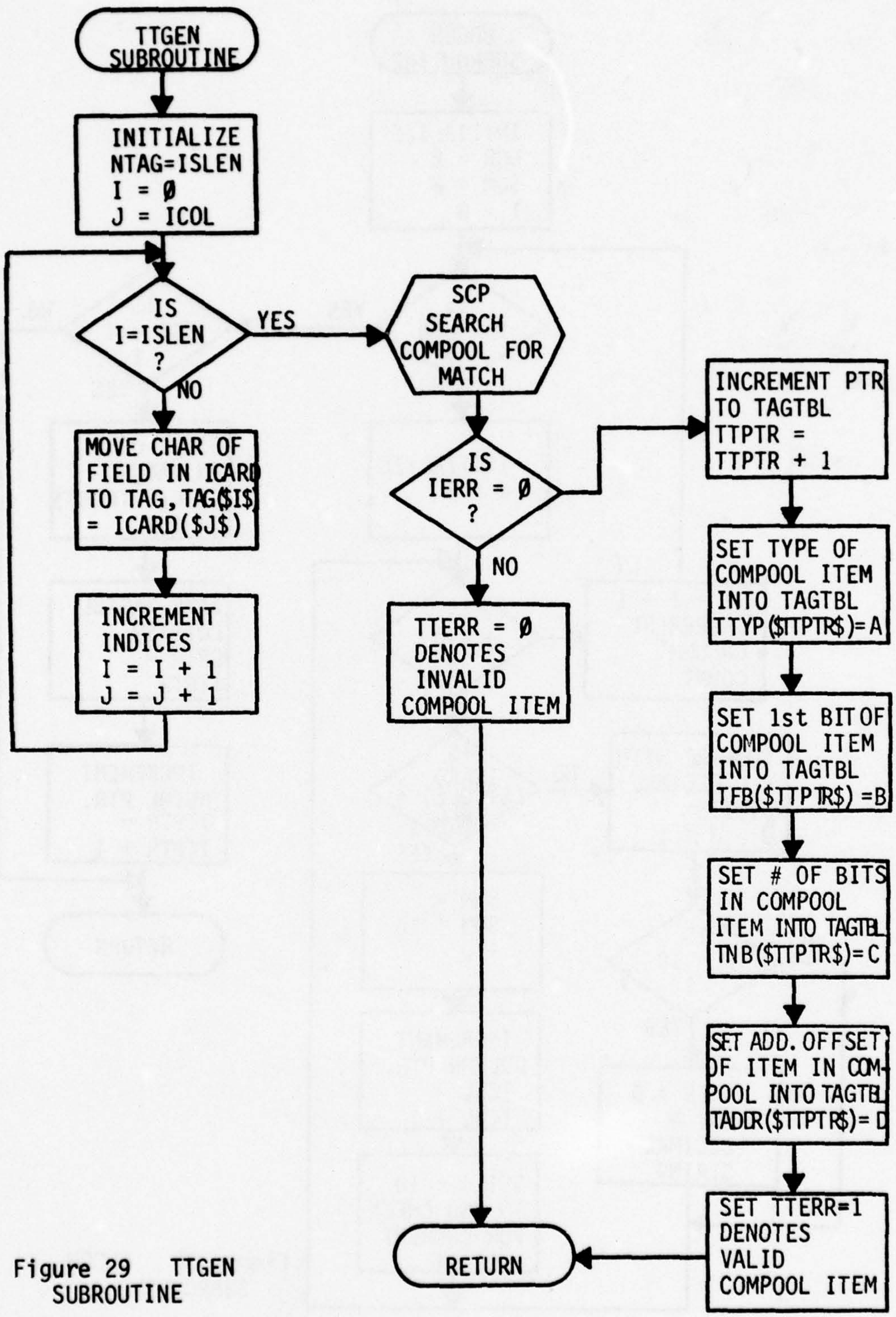


Figure 29 TTGEN SUBROUTINE

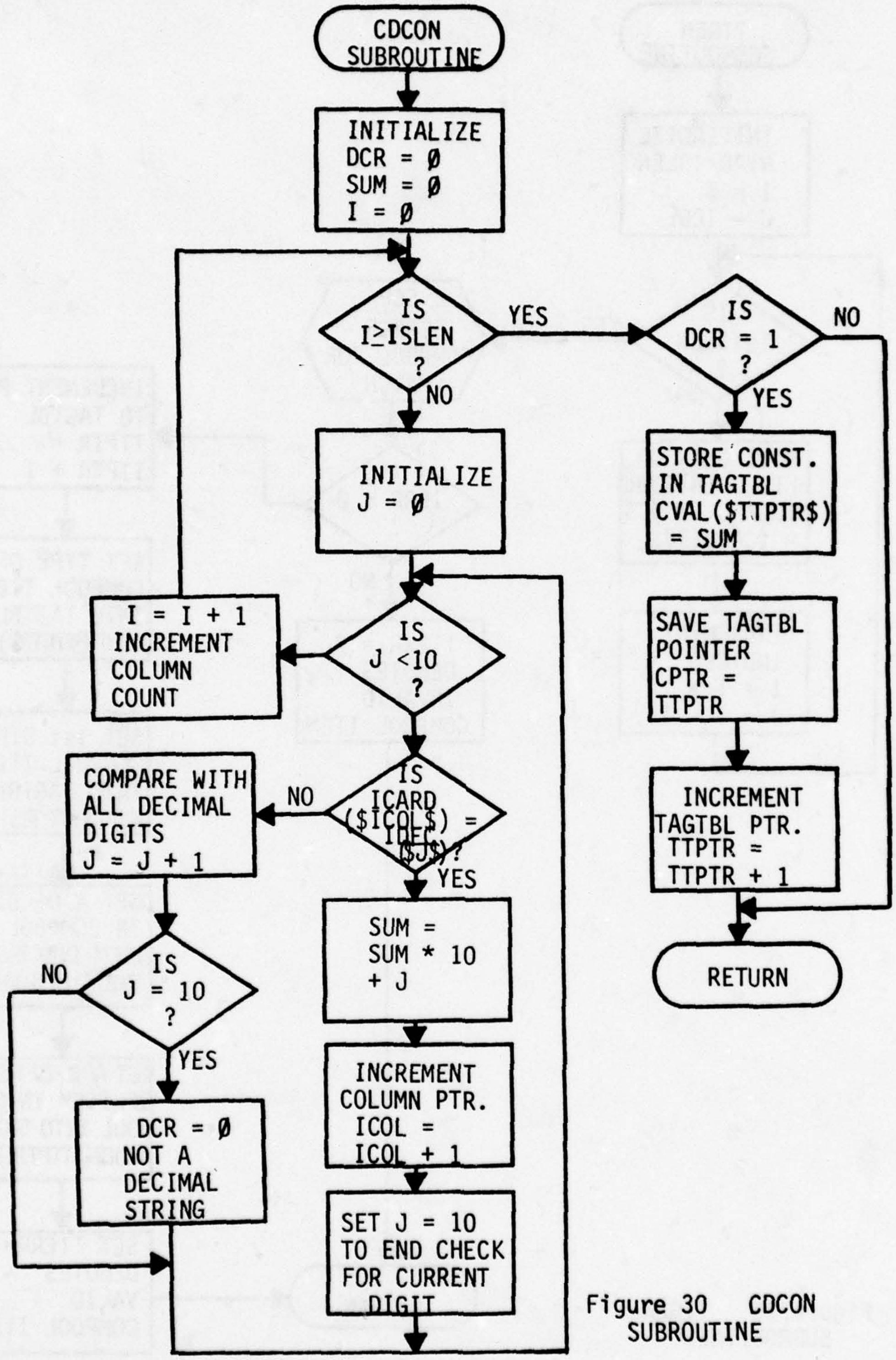


Figure 30 CDCON SUBROUTINE

Thus, each field of the command line fits into one of the following categories; keyword, compool item, decimal constant or character string (alphanumeric) constant. After the whole command line has been processed in this manner, one last call is made to POLGN to put the reverse Polish code in its final form before execution.

5.7.3 Code Execution. This section deals with the code execution routine, CCXEC, which is described in Paragraph 5.4.6. In addition, CCXEC also carries out the functions of error reporting discussed in Sections 5.4.4 and 5.4.5, although they are not fully implemented at this time, as well as the SEND RESULTS'TO'CONSOLE function dealt with in 5.4.7. A detailed flow of CCXEC is shown in Figure 31.

First, the subject of the command, event or report, is used to set the disk sector base and the record count for the code execution loop. Each record is read off of the disk into memory and the RPN string is executed using that record to yield a true or false result. If true, the proper routine is called, determined by the verb type MVTYP, to perform the desired operation on the current record. The next disk record is read in and the same sequence is repeated until all records have been read.

The actual execution of the RPN string takes place as follows. Each entry in the RPN string is handled separately. A check is made to see if the entry is an operator, which is denoted by preceding the keyword number on the stack with an escape character. Once the existence of the operator is determined, the keyword number is used to identify the correct routine to carry out the appropriate operation. All necessary operands have already been placed on the working stack by nature of the RPN string and the result of the operation is returned to that stack.

In a similar manner, the decimal constants are denoted by an '@' character which is followed by a pointer to the actual constant in the TAGTBL. The constant is placed on the working stack and control proceeds to the next entry on the RPN string.

For a character string, a slash character precedes a pointer to an entry in the TAGTBL which contains the length of the string and its starting column in the command line in the table ICARD. The data is placed on the working stack and a flag indicate string mode is set. This ensures proper handling of the string during its operation. Control then moves on to the next RPN entry.

If none of the special characters are recognized, the entry is just a pointer to the TAGTBL, which contains the attributes of a compool item. The attributes are used to place that compool item onto the working stack. Control again returns to handle the next entry on the RPN stack.

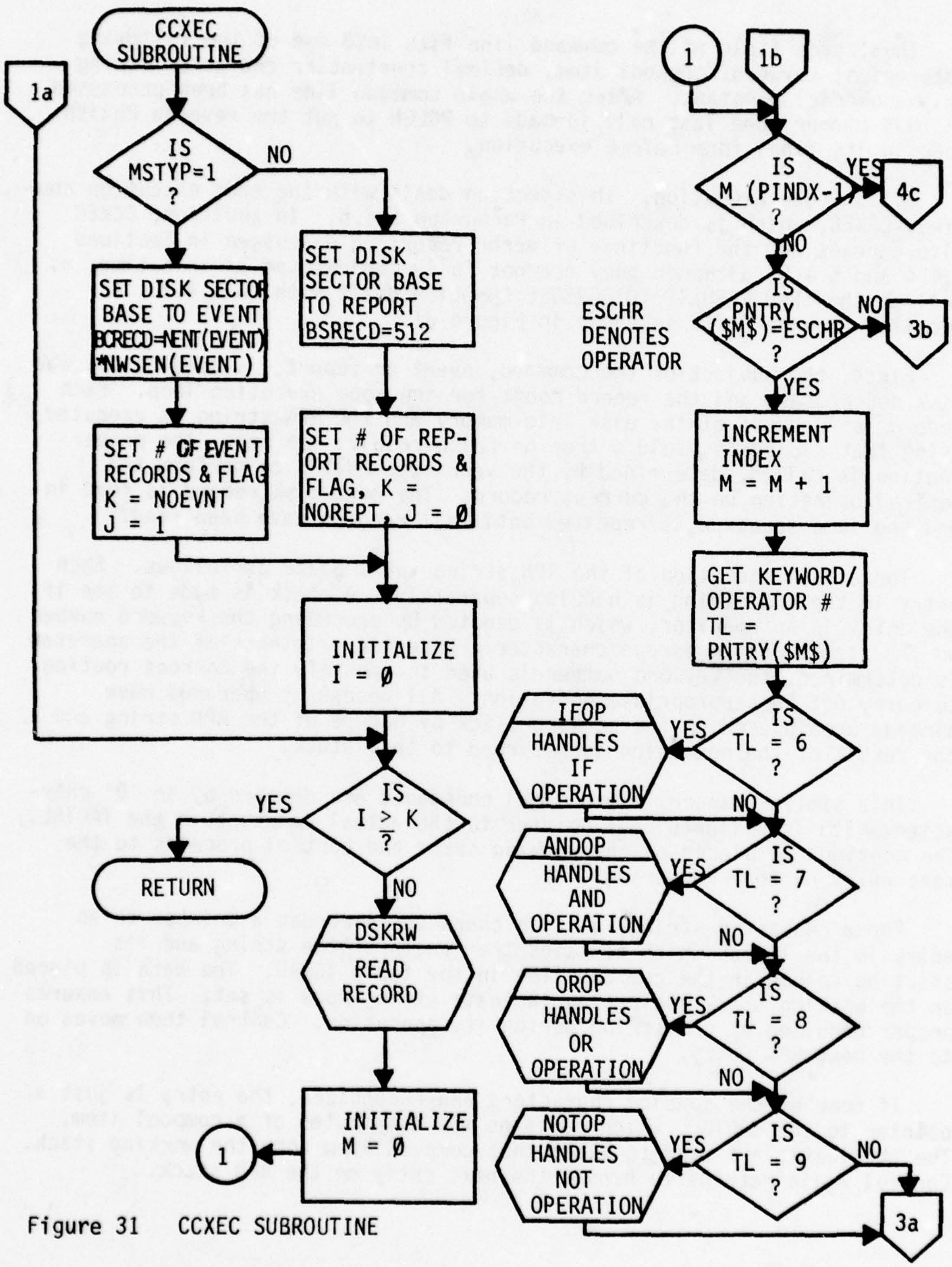


Figure 31 CCXEC SUBROUTINE

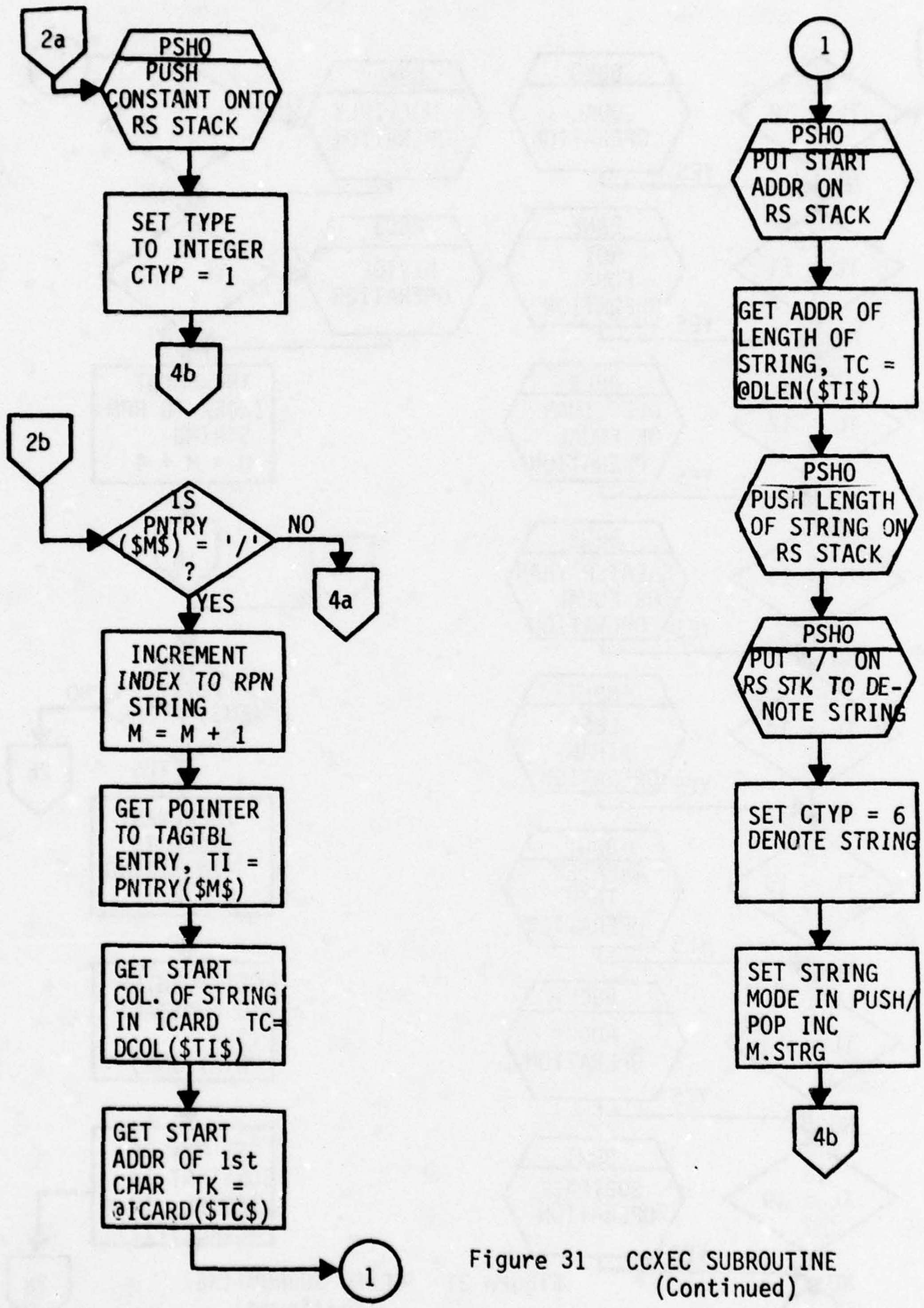


Figure 31 CCXEC SUBROUTINE (Continued)

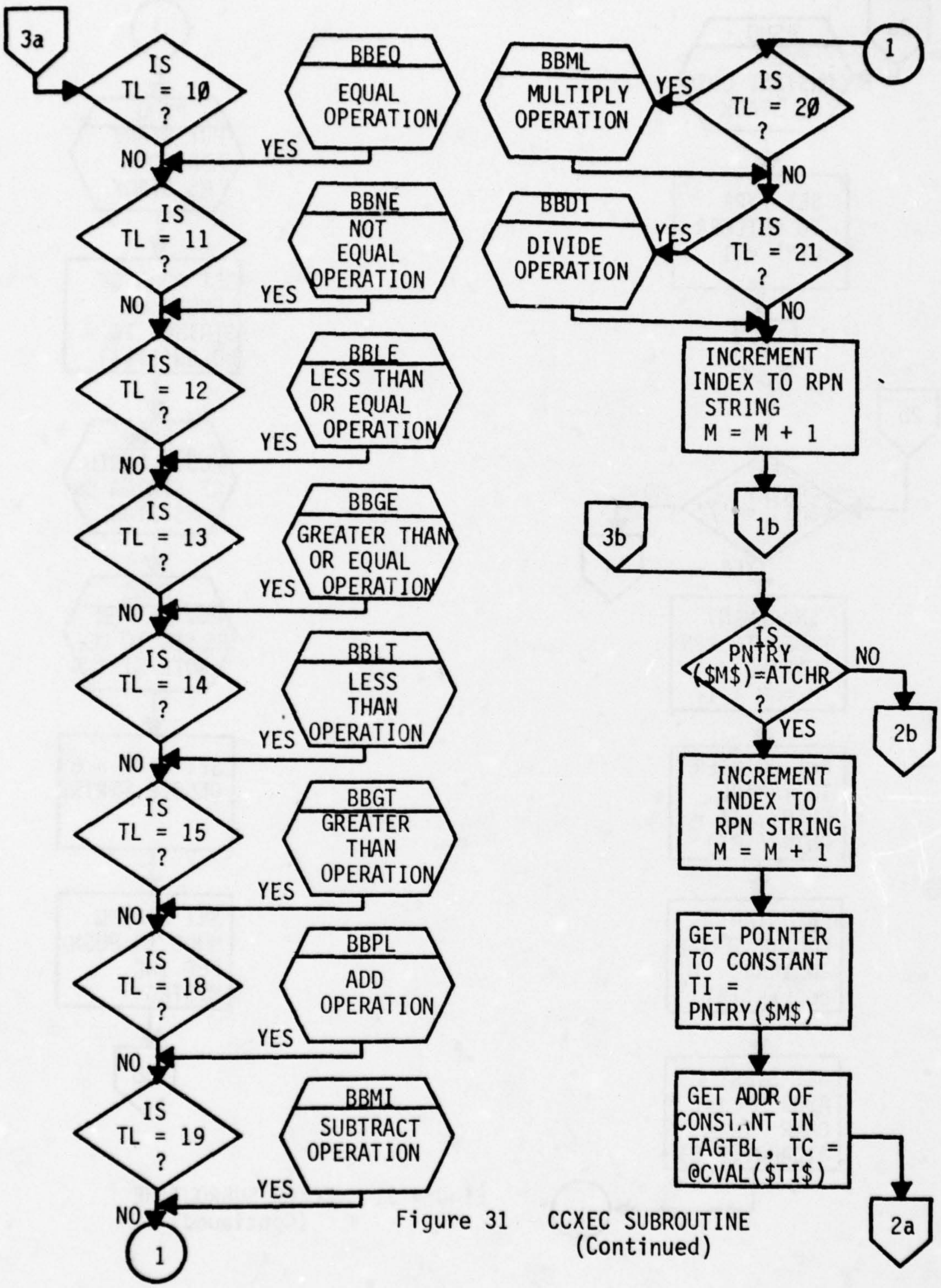


Figure 31 CCXEC SUBROUTINE (Continued)

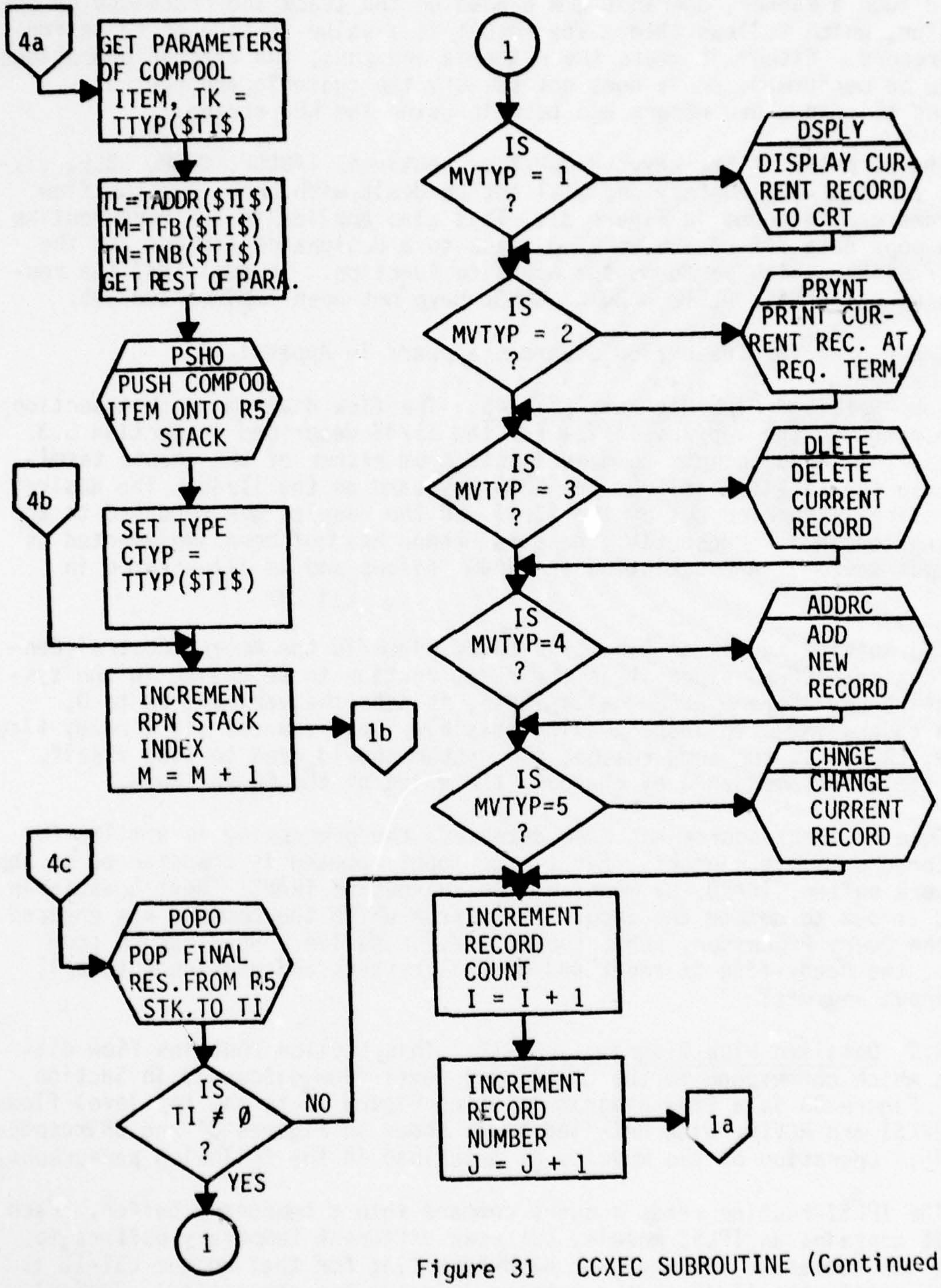


Figure 31 CCXEC SUBROUTINE (Continued)

In such a manner, operands are placed on the stack and processed by the operator, which follows them. The result is a value of true or false for each record. Either it meets the criteria and thus, the desired operations should be performed, or it does not satisfy the criteria and control returns to read a new record and test it using the RPN string.

The purposes of the keyword/operator routines, (ANDOP, OROP, BBLE, ... etc.) are self-explanatory and will not be dealt with here, but the flow diagrams can be found in Figure 31. This also applies to the POPO routine which pops data off of the working stack to a designated address and the PSHO routine, which performs the opposite function. In addition, the routines DSPLY, PRMNT, DLETE, ADDRC, CHNGE have not been implemented yet.

The coding for these flow diagrams appears in Appendix A.

5.8 Detailed Flow Diagrams - 11/45. The flow diagram in this section corresponds to the top level flow for the 11/45 described in Section 5.3. The routine INP45 accepts command inputs from either of the remote terminals via the LSI-11's or from the LA36 keyboard on the 11/45. The desired processing is carried out by the 11/45 and the results are returned to the calling terminal. Presently, the card reader has not been implemented as an input source. A description of INP45 follows and is illustrated in Figure 32.

The initialization of the 11/45 takes place in the Access Control Center routine, ACCPM, since it is the first routine to be called in the system start-up. Before ACCPM calls INP45, it sets the variable GO to 0, which causes INP45 to loop, polling possible input sources via a ready flag table, LRDY. If for some reason, the system should need to stop itself, this can be accomplished by changing the value of the GO variable.

Once an input source has been detected, the processing is similar for all three possible sources. First, the input command is transferred to the RPN work buffer, ICARD, by means of the subroutine TRANC. Next a variable INSRC is set to denote the input source from which the command was entered and the Query Processor, CCNTL subroutine, is called. Upon return from CCNTL, the ready-flag is reset and control returns and continues to poll the input sources.

5.9 Detailed Flow Diagram - LSI-11. This section contains flow diagrams which correspond to the LSI-11 top level flow discussed in Section 4.2. Figure 33 is a flow diagram relating Figure 18 to the top level flows for IPLSI and RCVIP, with detailed flows shown in Figures 34 and 35 respectively. Operation of the modules is described in the following paragraphs.

The IPLSI routine reads a query command into a temporary buffer. Each LSI-11 contains an IPLSI module, but uses different temporary buffers to store the command. The ready or handshake flag for that unique LSI-11 is set to notify the 11/45 that a command is ready for processing. Control

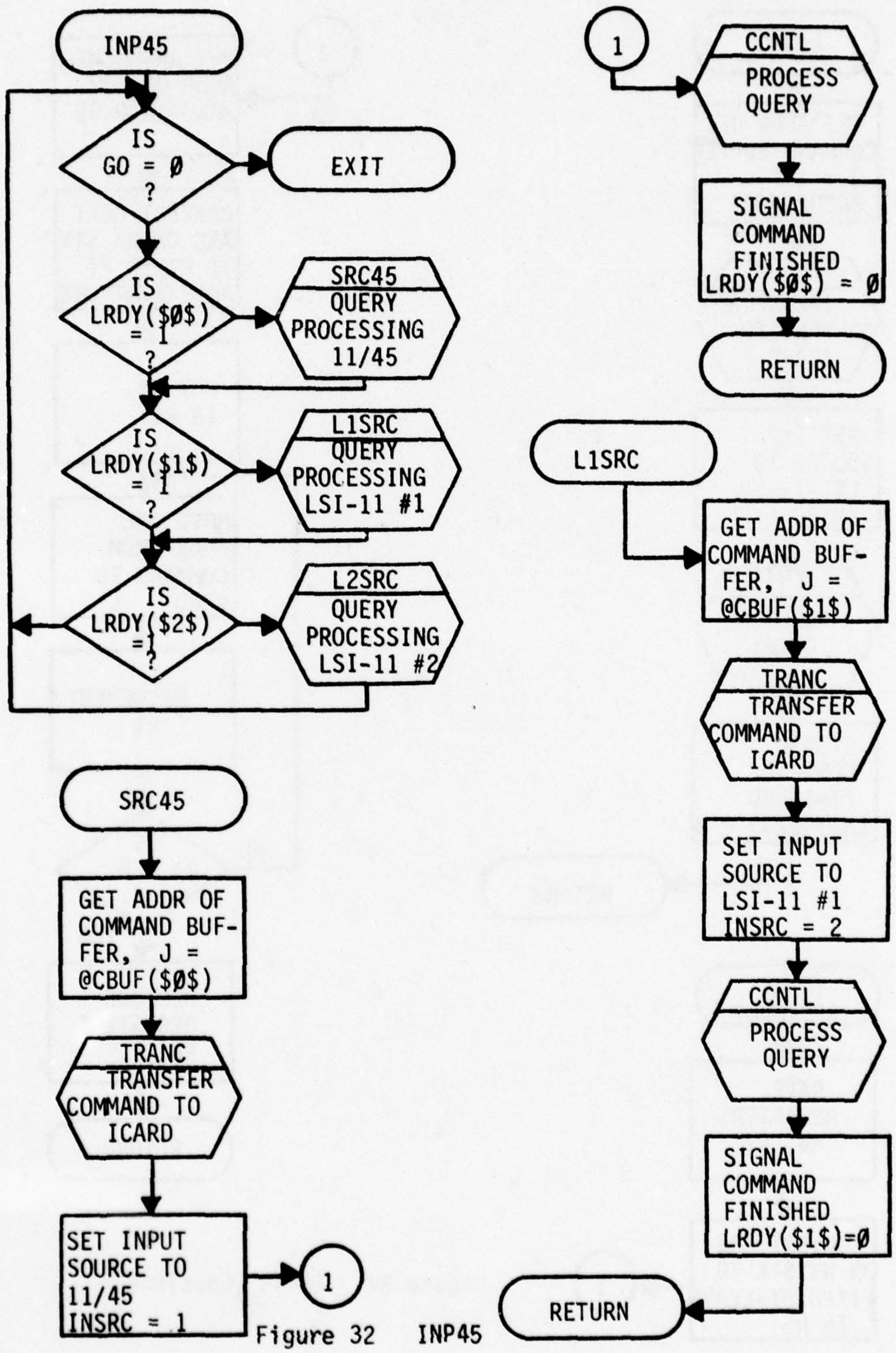


Figure 32 INP45

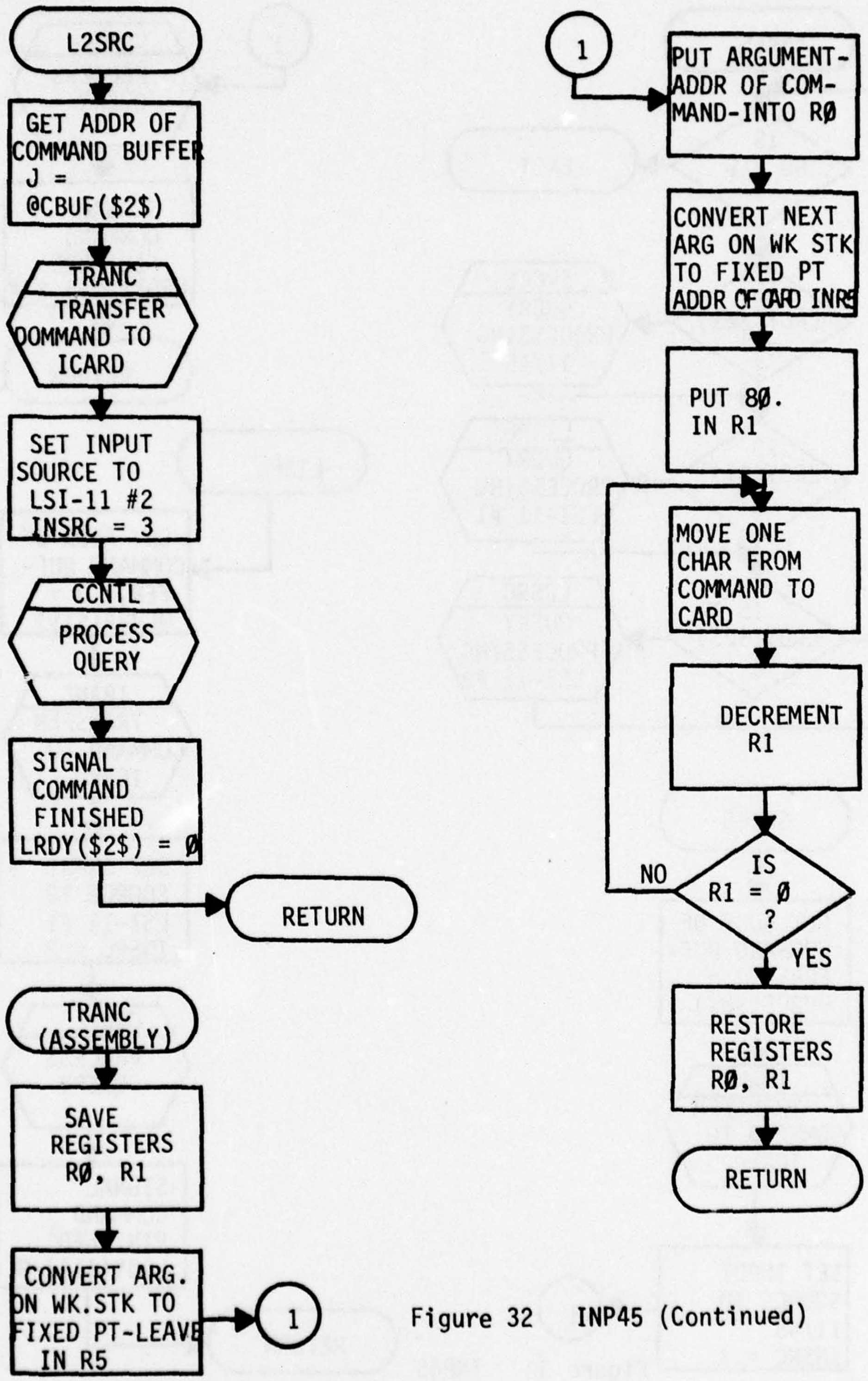


Figure 32 INP45 (Continued)

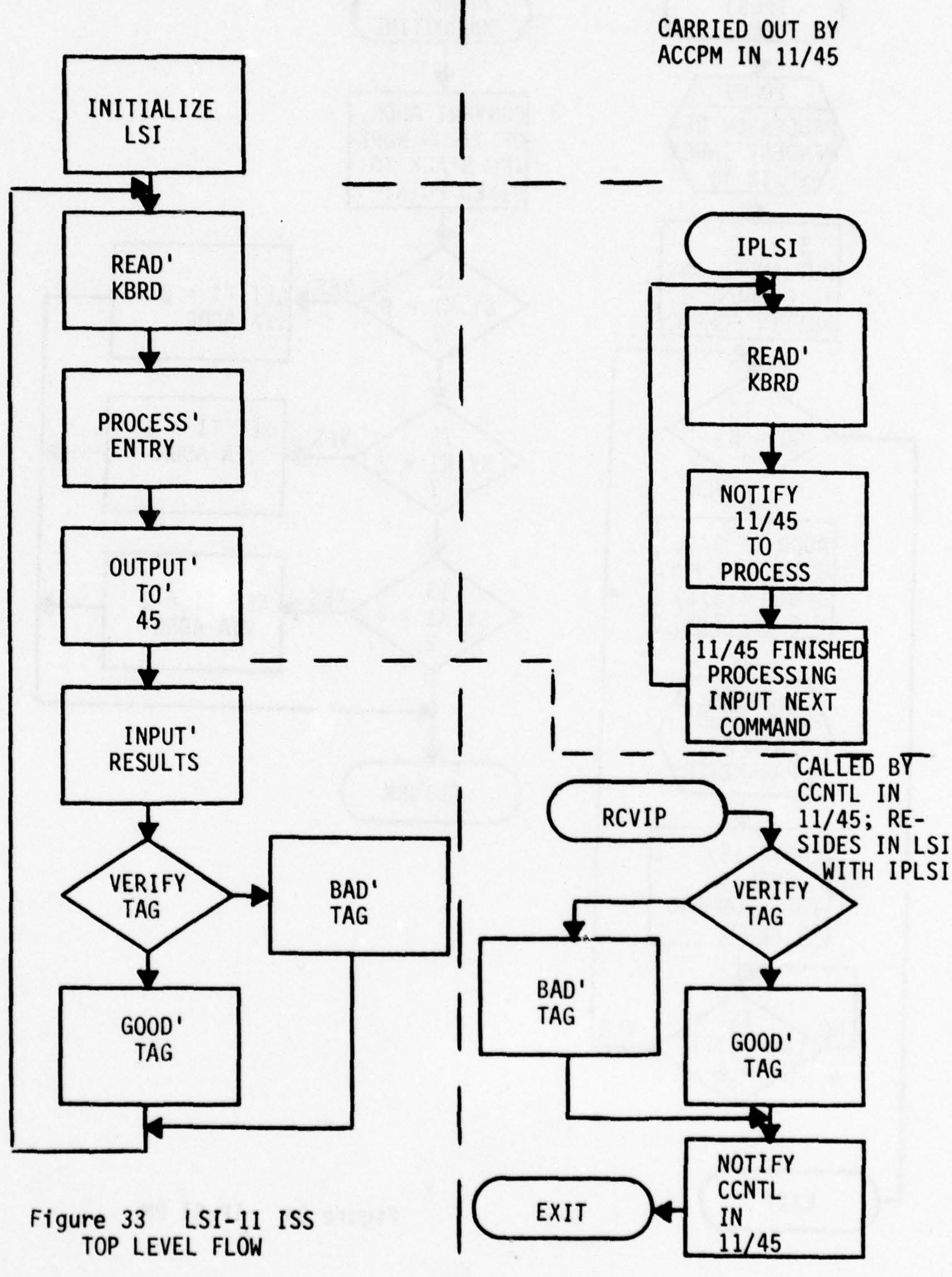


Figure 33 LSI-11 ISS TOP LEVEL FLOW

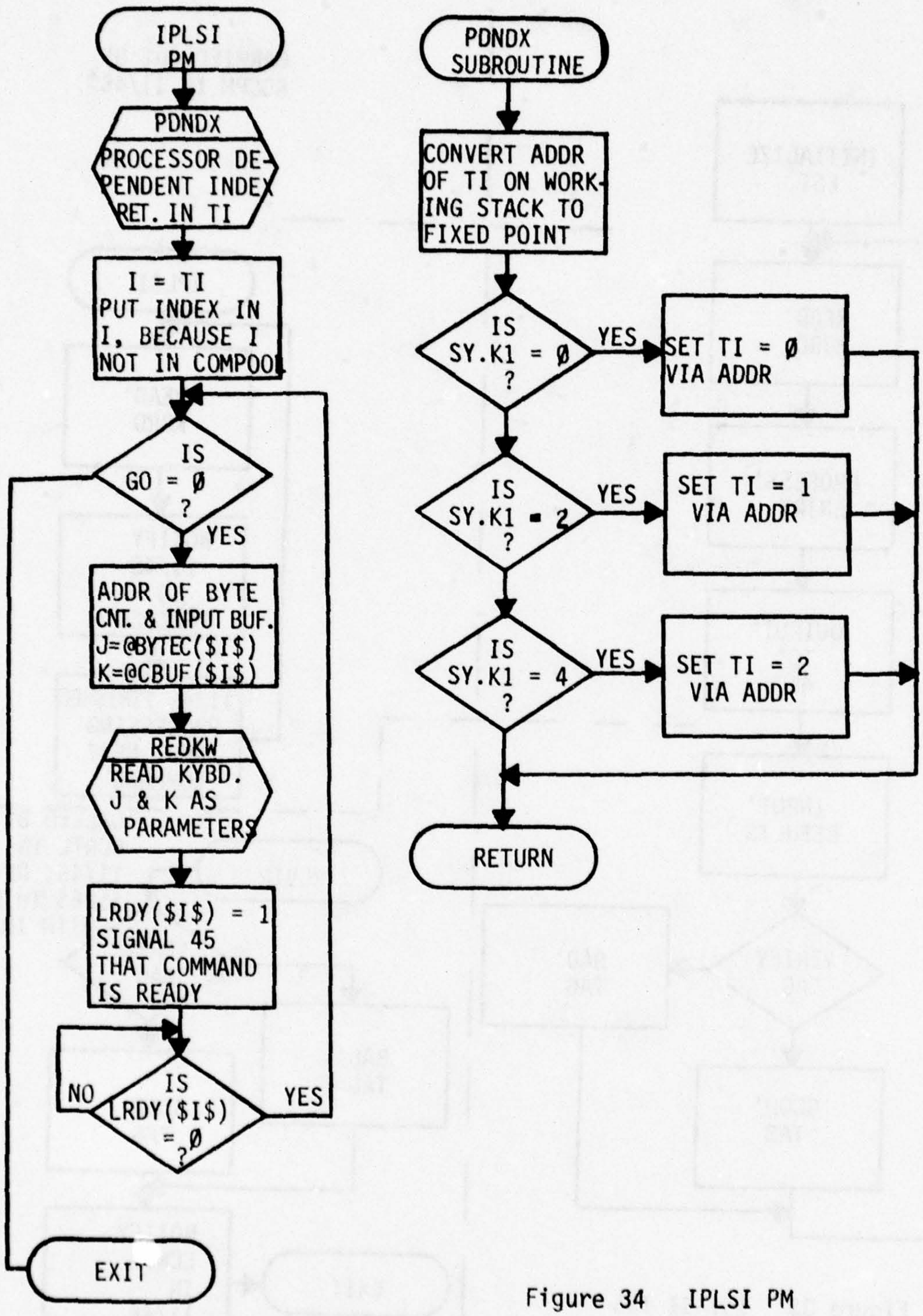


Figure 34 IPLSI PM

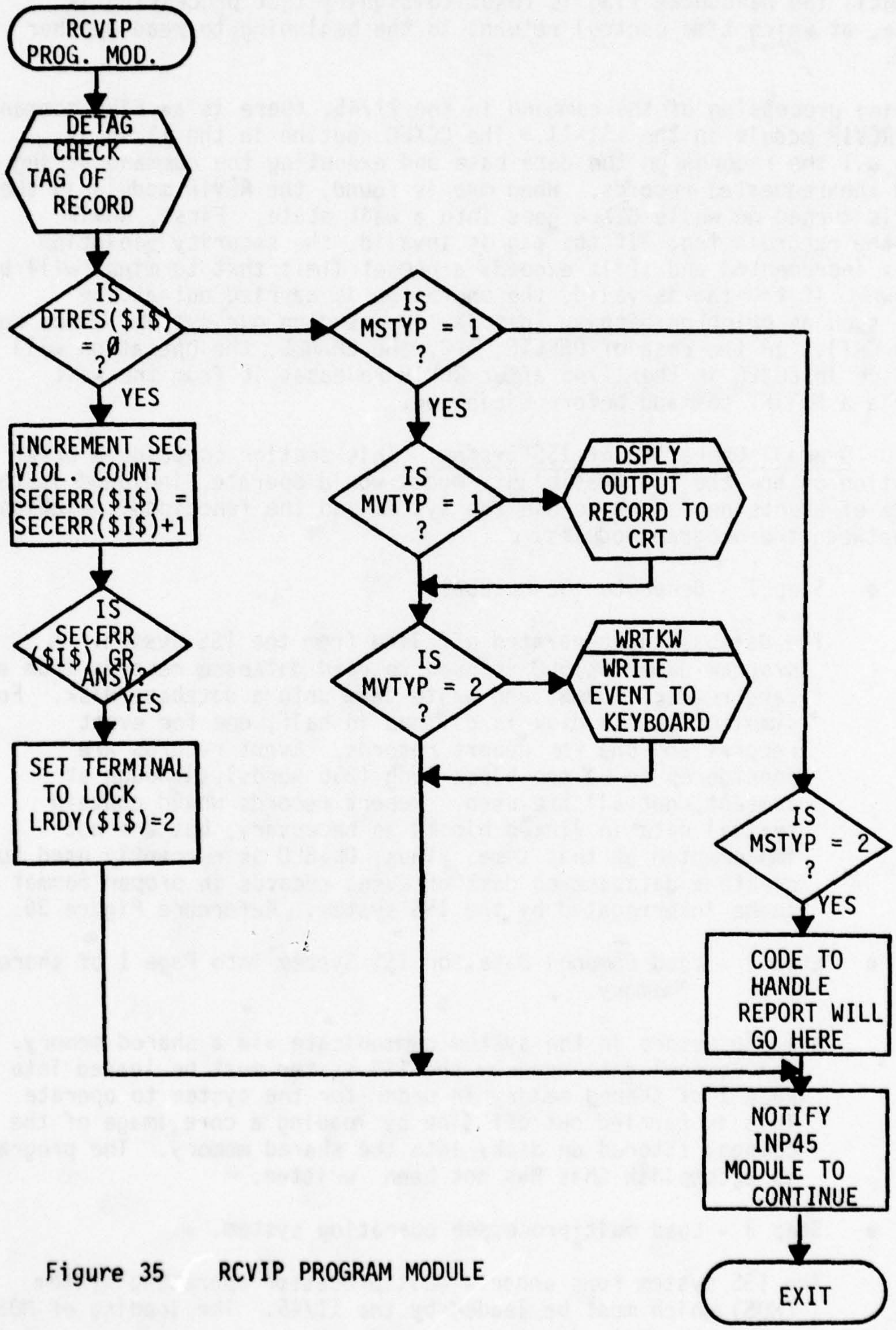


Figure 35 RCVIP PROGRAM MODULE

loops until the handshake flag is reset to signify that processing is complete, at which time control returns to the beginning to read another command.

During processing of the command in the 11/45, there is an EXEC command to the RCVIP module in the LSI-11. The CCXEC routine in the 11/45 is reading all the records on the data base and executing the command string to find the requested records. When one is found, the RCVIP module in the LSI-11 is turned on while CCXEC goes into a wait state. First, RCVIP checks the record's tag. If the tag is invalid, the security violation count is incremented and if it exceeds a preset limit that terminal will be shut down. If the tag is valid, the operation is carried out at the LSI-11, such as print or display (display not used on our system, since we have no CRT). In the case of DELETE, ADD, and CHANGE, the operation will take place in CCXEC in the 11/45 after RCVIP releases it from the wait state via a NOTIFY command before finishing.

5.10 Overall Operation of ISS System. This section contains a brief description of how the ISS feasibility model would operate, included is the sequence of events necessary to use the system and the functional relationships between the program modules.

- Step 1 - Generate the database.

  The database is generated off line from the ISS system. A program named DK1BLD is used to read database records from a card reader, format and write them onto a database disk. For simplicity, the disk is divided in half, one for event records and one for report records. Event records are considered to be one block long (256 words) although at present, not all are used. Report records would contain textual data in linked blocks as necessary, but are not implemented at this time. Thus, DK1BLD is currently used to create a database on disk of event records in proper format to be interrogated by the ISS system. Reference Figure 36.

- Step 2 - Load Compool Data for ISS System into Page 1 of shared memory.

  All processors in the system communicate via a shared memory. The Compool data used by the ISS system must be loaded into Page 1 of shared memory in order for the system to operate. This is carried out off line by loading a core image of the compool (stored on disk) into the shared memory. The program to accomplish this has not been written.

- Step 3 - Load multiprocessor operating system.

  The ISS system runs under a multiprocessor operating system (MOS) which must be loaded by the 11/45. The loading of MOS

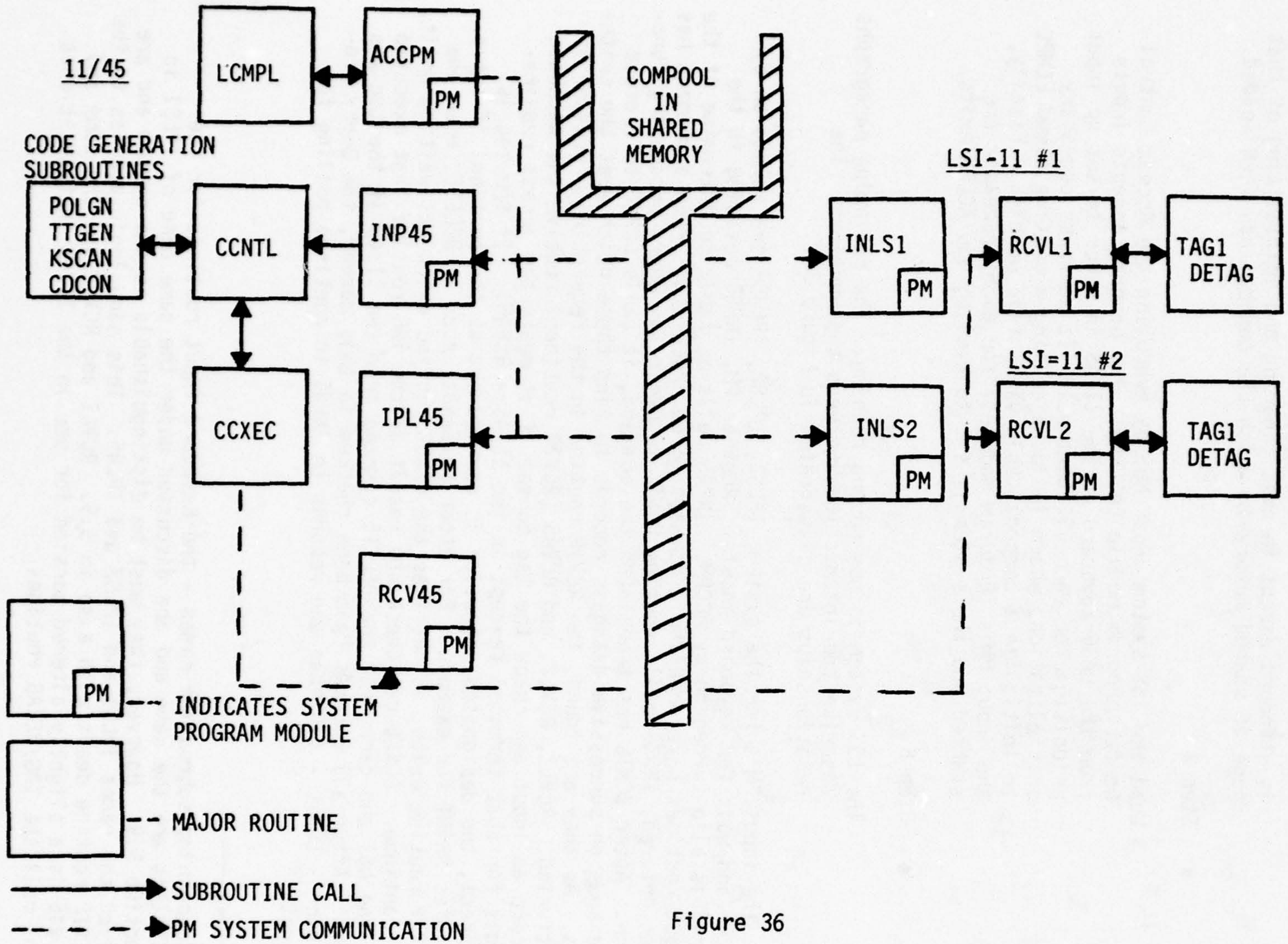

11/45
LCMPL
ACCPM
PM
CODE GENERATION SUBROUTINES
POLGN
TTGEN
KSCAN
CDCON
CCNTL
INP45
PM
CCXEC
IPL45
PM
RCV45
PM
COMPOOL IN SHARED MEMORY
LSI-11 #1
INLS1
PM
RCVL1
PM
TAG1 DETAG
LSI=11 #2
INLS2
PM
RCVL2
PM
TAG1 DETAG
PM
INDICATES SYSTEM PROGRAM MODULE
MAJOR ROUTINE
SUBROUTINE CALL
PM SYSTEM COMMUNICATION

Figure 36

is standard except to omit clearing and initializing of that page of shared memory in which the compool has been loaded.

- Step 4

Load the ISS system under MOS by executing the Access Control Center program module (ACCPM). The program accepts inputs from the LA36 keyboard on the 11/45 in order to set up input priorities for each LSI terminal. It performs necessary initialization, which includes calling a routine named LCMPL to initialize a compool table via a file on disk. Finally, the input PM's (Program Modules) for each processor are started and the system is set to cycling and ACCPM exits.

- Step 5

The ISS system is now up and running. The following paragraphs describe the interactions between the routines. The relationships are illustrated in Figure 36.

The input PM's for the system, IPLS1, IPLS2, IPL45* are turned on by ACCPM and wait for keyboard inputs. Another PM, INP45 residing in the 11/45 is also turned on by ACCPM. INP45 polls a table to determine if the input routines have received a command to be processed. When a command has been entered, INP45 calls routine CCNTL to begin the Query processing function. After CCNTL has translated the command, it calls CCXEC to execute the code on successive database records to find those which meet the criteria. As they are found, the RCVIP routine in the requesting processor is activated. RCVL1, RCVL2, and RCV45 (RCVIP routines) receive the database record as input and check the tag to see if it meets the access requirements for that terminal (except in the 11/45 - RCV45). If the tag is correct, the designated operation is carried out at the terminal if appropriate, PRINT for example. For other commands, such as DELETE, when the RCVIP routine exits, it notifies the CCXEC routine which was waiting on it, to continue. CCXEC can check the result of the TAG routine that executed in the LSI and carry out the DELETE command from the 11/45 if the tag was good. After all records have been checked in this manner, the Query Processor - CCNTL - finishes and returns to INP45 to continue polling for inputs.

---

*Concerning subroutine names - the keyboard input routines for the LSI-11's are the same and are discussed under the same name of IPLSI in Section 5.9. However, they must be distinguishable to the system and are given the names IPLS1 and IPLS2 and IPL45. This same logic applies to the PCVIP routine dealt with also in 5.9. RCVL1 and RCVL2 are the same and RCV45 is a slightly altered version for use in the 11/45 in that it does not call the TAG/DETAG routines.

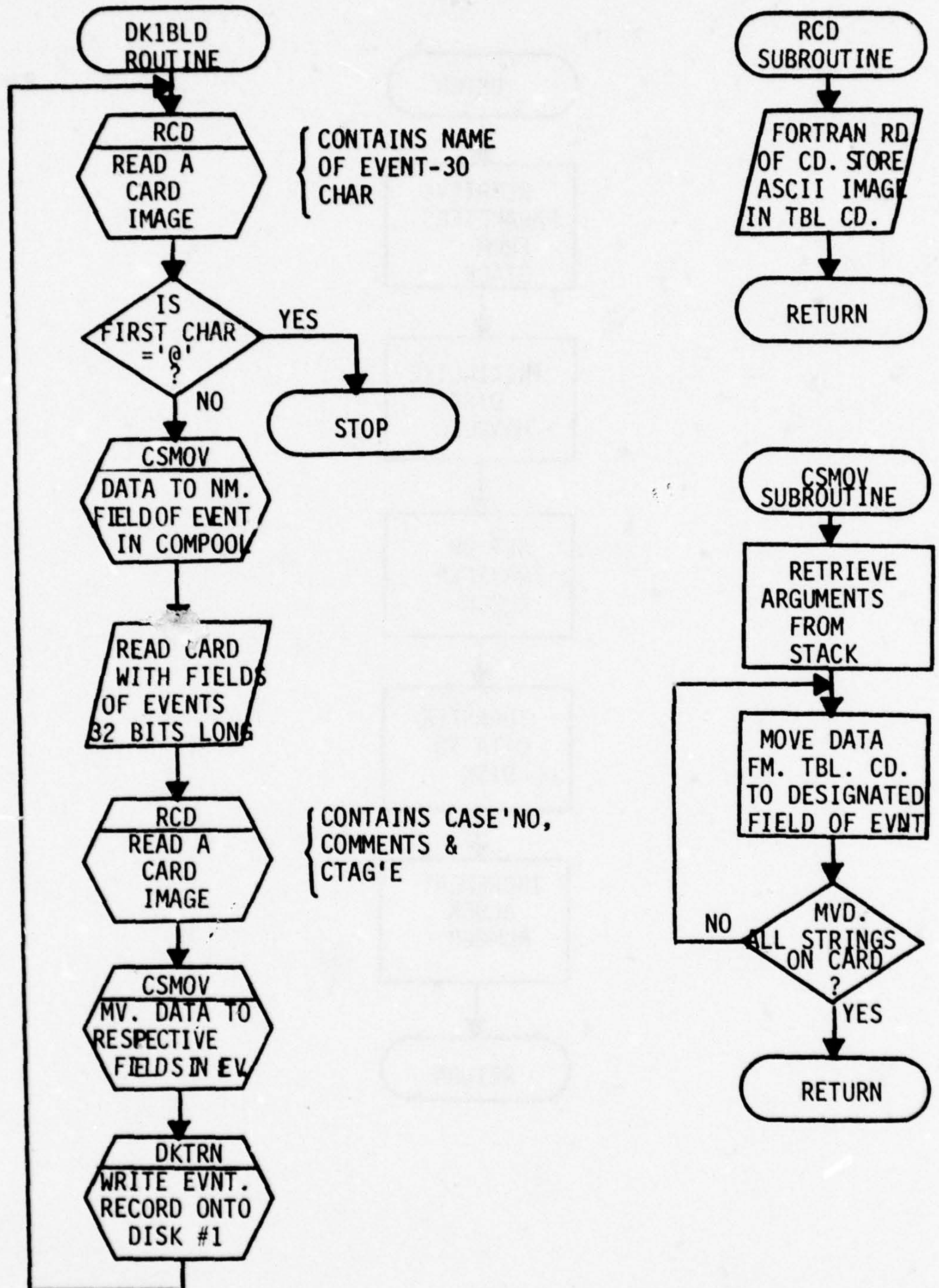


Figure 37 DK1BLD ROUTINE

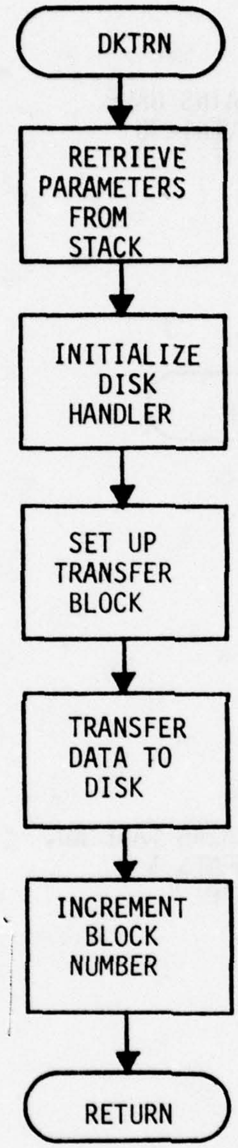


Figure 37 DKIBLD ROUTINE (Continued)

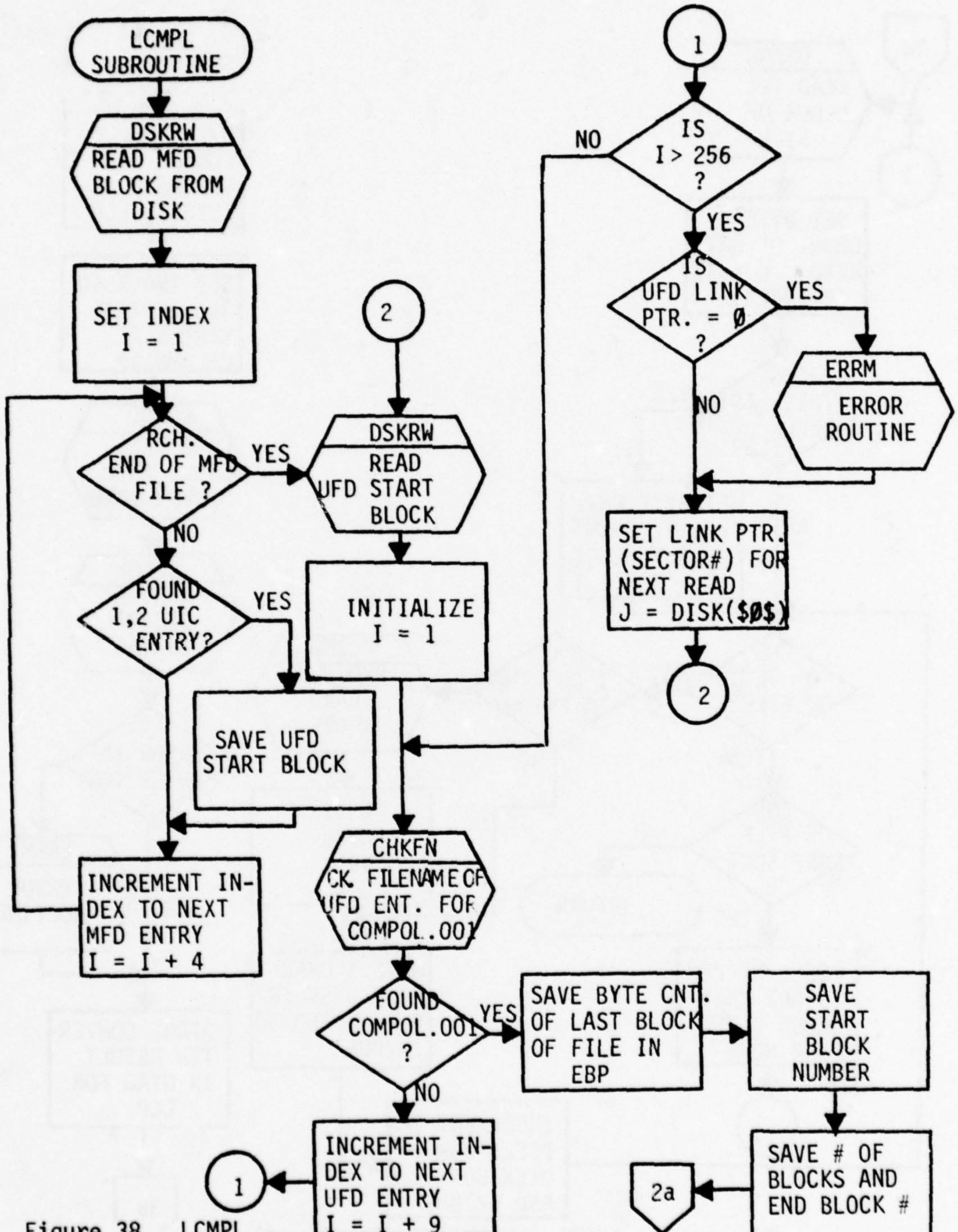


Figure 38 LCMPL SUBROUTINE

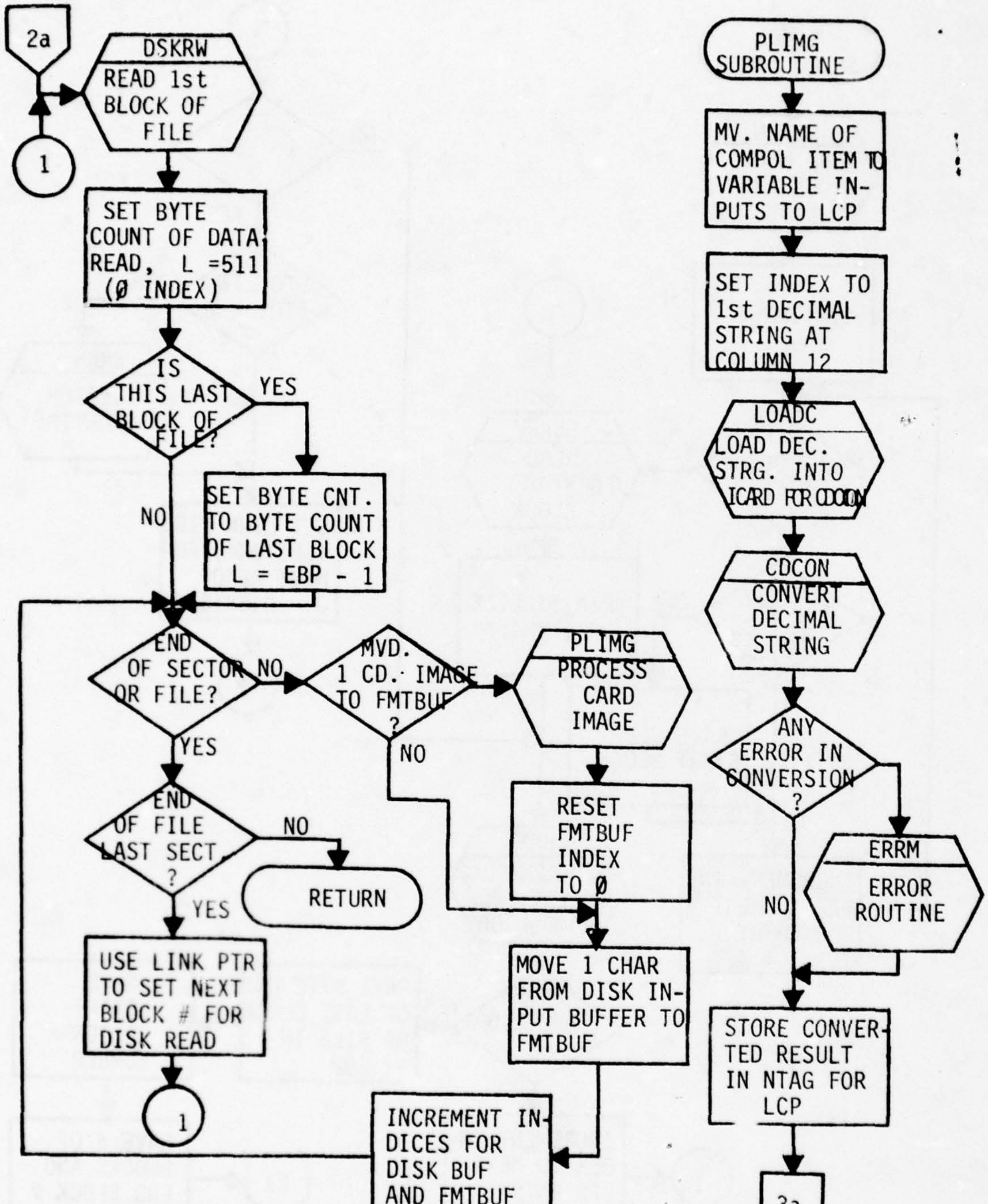


Figure 38 LCMPL SUBROUTINE (Continued)

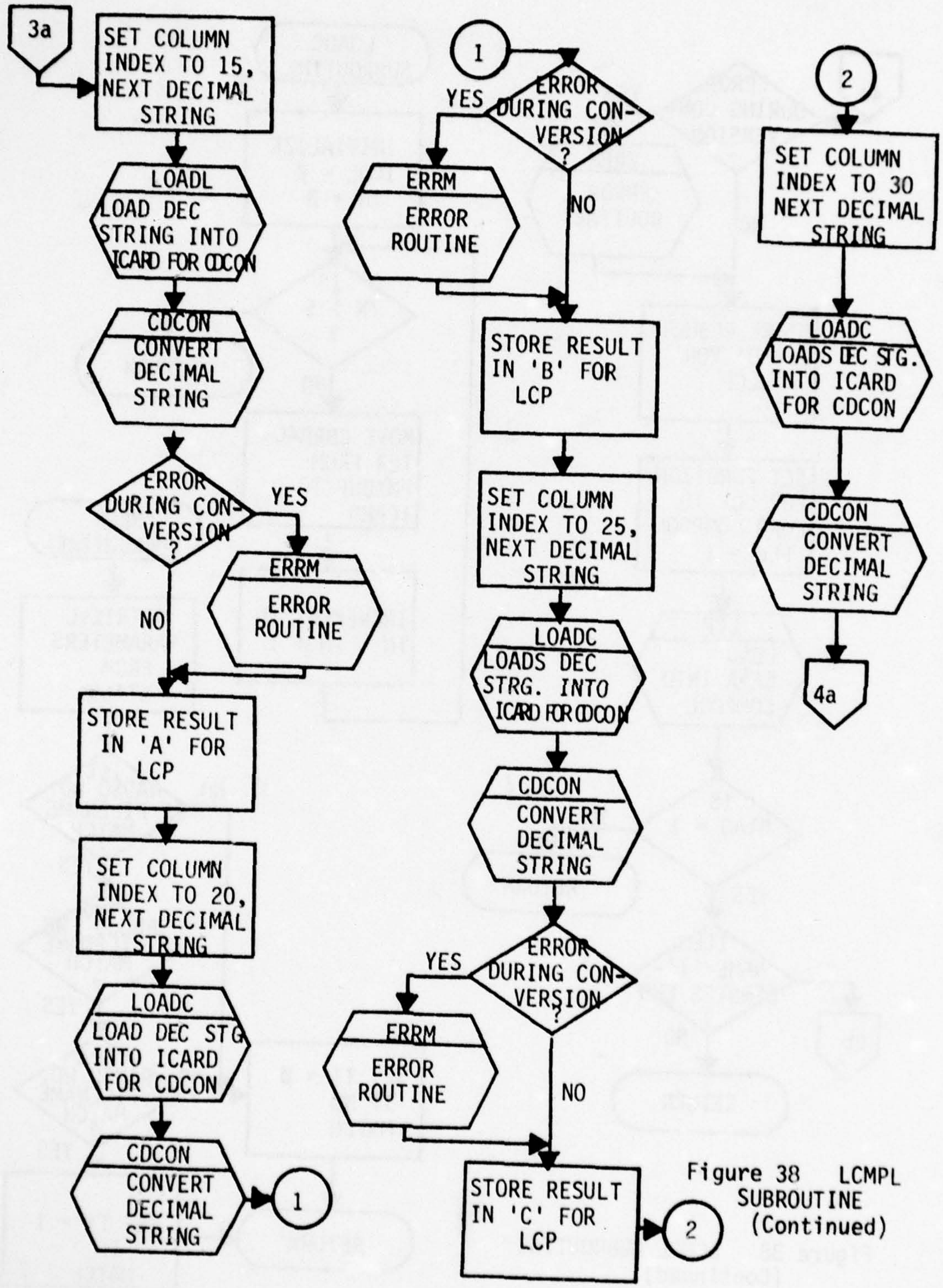


Figure 38 LCMPL SUBROUTINE (Continued)

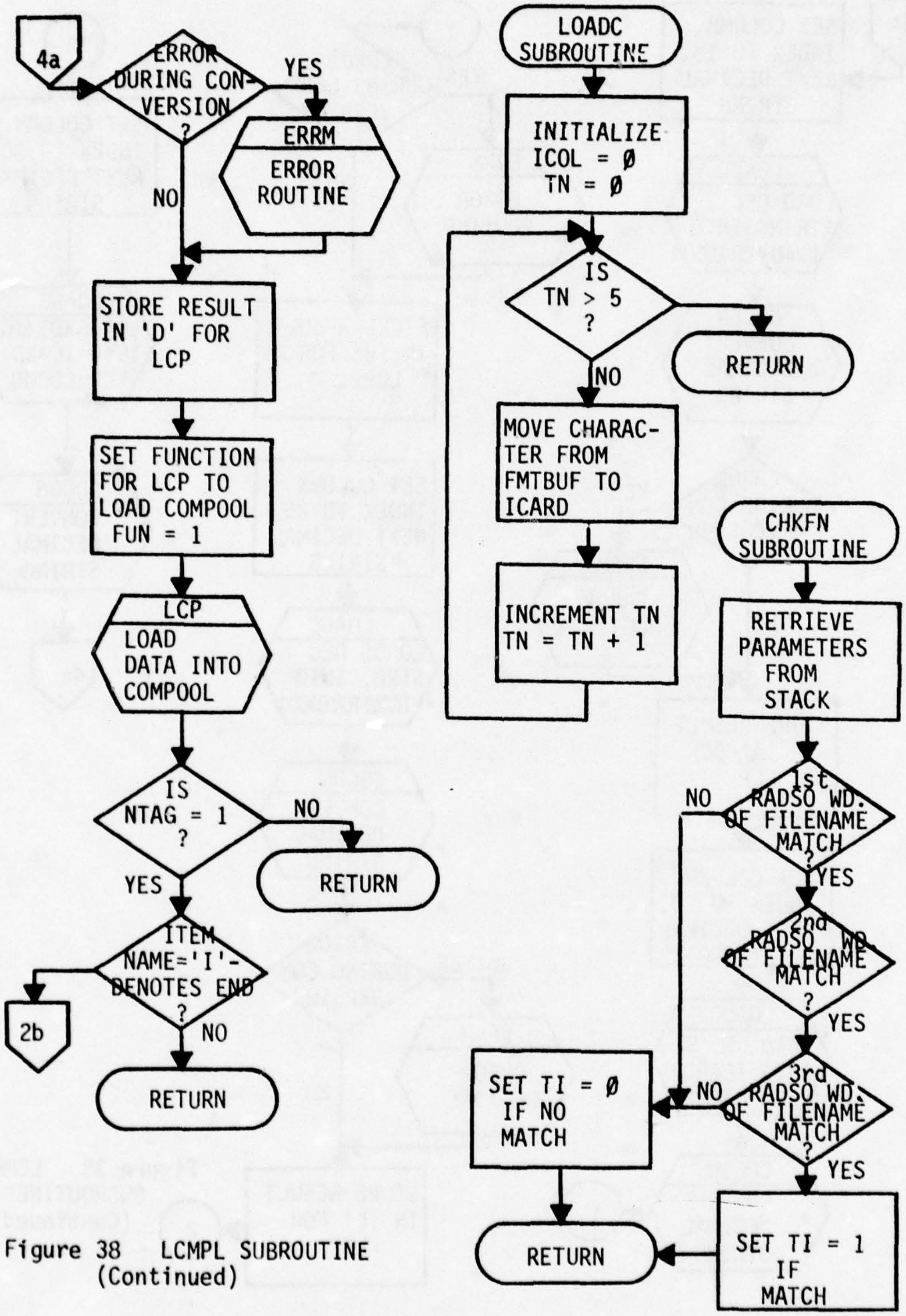


Figure 38 LCMPL SUBROUTINE (Continued)

SECTION 6.0

RED-BLACK MULTIPROCESSING

6.0 RED-BLACK MULTIPROCESSING

6.1 Introduction. This section presents a promising solution to the Secure Data Base design problem. The solution, called a RED-BLACK Multiprocessing System, is demonstrably secure, conceptually simple, and quite easy to implement. Security is provided mainly by complete physical separation of data processing functions. All processing of classified data is done on one processor called the Red Processor, while all processing of unclassified data is done on another processor called the Black Processor. There is no sharing of main memory between the two processors, and communications between the Red and Black Systems are highly constrained to prohibit any classified data leakage.

Some terminals which are connected to the Black Processor System may be used for interactive processing of either classified or unclassified data. Other terminals are connected to the Black Processor System, which may be used for interactive processing of unclassified data only. (Reference Figure 39.) In the former case, requests for classified data processing are forwarded to the Red Processor by the Black, and the classified output is returned from the Red via the Black in encrypted form with decryption performed at the terminal. Thus, the Black Processor System may function in the handling of encrypted classified data, but it never has access to raw or processed classified data in plain text form.

The remainder of this section contains the following:

- Paragraph 6.2 - The threat environment is defined together with other relevant assumptions.
- Paragraph 6.3 - System design goals described.
- Paragraph 6.4 - Proposed system design description.
- Paragraph 6.5 - Proposed design security against various elements of the assumed threat environment.
- Paragraph 6.6 - Simple performance/cost evaluation.
- Paragraph 6.7 - Summary and conclusions.
- Paragraph 6.8 - References.

6.2 Threat Environment and Related Assumptions. This section summarizes the threat environment and related assumptions necessary for the System Security Evaluation discussed in Paragraph 6.4. The threat environment description consists mainly of a brief catalog of persons assumed to be potential enemy agents, together with what they are assumed to know and penetration tools at their disposal. The related assumptions pertain to persons who are assumed to not be potential enemy agents, and to the events which are assumed to be not possible. A general block diagram of the proposed RED-BLACK Multiprocessing Scheme is shown in Figure 39 to support the discussion.

6.2.1 Personnel Assumptions. Table 23 identifies all classes of persons of interest. Persons considered as potential enemy agents are referenced as (potential) agents, while those assumed to be absolutely trustworthy are referred to as good personnel. Potential agents are

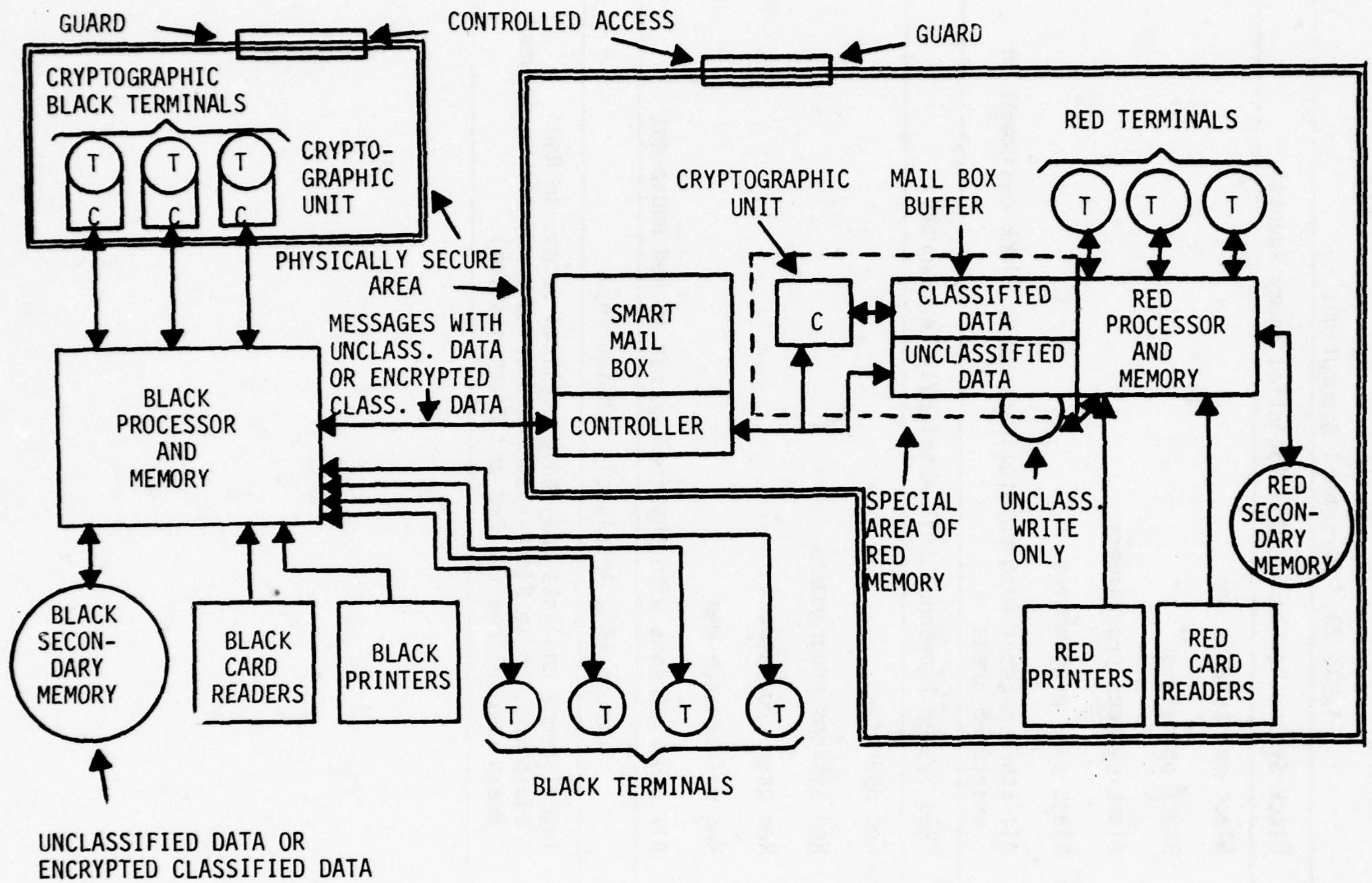


Figure 39 RED/BLACK MULTIPROCESSING SYSTEM GENERAL BLOCK DIAGRAM

Table 23. Personnel Assumptions

Black System Personnel (All Potential Enemy Agents)
• Black maintenance men • Black operators • Black systems programmers • Black user programmers • All other persons with physical access to Black equipment in unsecured areas
Red System Personnel (All Absolutely Trustworthy)
• Red operators • Red systems programmers • Red user programmers • Red maintenance men • All other persons with physical access to Red equipment
Others (All Absolutely Trustworthy)
• Intelligence analysts (may have physical access to Red equipment, or to Black-system-connected terminals in secured areas remote from the Red and Black systems)

assumed to be highly intelligent, resourceful, and have access to all knowledge in possession of the enemy intelligence community; also, they are able to use all applicable enemy intelligence technology. Specifically, the potential agents are assumed to have complete knowledge of the hardware and software of both Red and Black systems. Since Black operators and systems programmers are potential agents, they are assumed to be capable of undertaking their penetration objectives under the guise of normal work activities. Because some Black terminals can be located anywhere, bad black terminal users are assumed to be able to conduct penetration efforts without concern over discovery by physical surveillance.

On the other hand, it is logical to assume that certain other classes of persons must be inherently good. In this category belong all persons with physical access of any kind to Red equipment. (Some possible exceptions to this may exist, but they will not be discussed here.)

6.2.2 Equipment Assumptions. The main equipment assumptions are:

- The cryptographic scheme used in the cryptographic units has been subjected to intense cryptoanalysis, and is certifiably secure against any feasible code breaking efforts.*
- It is not feasible for any potential agent to determine a code key from physical or electronic examination of tamper-proof cryptographic Black terminals. (Harris designs and produces tamper-proof terminals for Secure Voice Communications.)
- Red system malfunctions are infrequent, and are fail-safe from a security standpoint.
- The Red system is in a physically and electronically secure area such that no potential agent can gain direct physical or electronic access to any equipment in the Red area (except for the smart mailbox, to which the potential agent has direct electronic access but not physical access).
- Black/Red System communications are solely by means of the smart mailbox as shown in Figure 39.
- There is no way for classified plain text data, processed or unprocessed to ever be placed in the smart mailbox, because of hardware controls in the Red System.

6.3 System Design Goals. It is desired that:

- The RED-BLACK Multiprocessing System be absolutely secure with respect to the threat environment identified in Paragraph 6.2.
- The system provide sufficient performance and capacity for processing all classified data and for storage of some classified data in plain text form.
- The system be easy to use;

*Such schemes are known to exist. It is assumed that one such scheme will be used.

- The system be reasonably cost-effective relative to a nonsecure system with similar performance, capacity, and ease of use.

6.4 System Design Description. Because the system design (as shown in Figure 39) is conceptually simple and involves the implementation of few new hardware or software techniques, it will not be described in full detail here. The basic idea is to:

- Enforce protection of classified data processing by complete physical isolation of the classified data processing function.
- Enforce protection of classified data transmission by using cryptographic techniques.
- Prevent classified data leakage by using a "smart mailbox" scheme to tighten control of RED-BLACK communications. This may be realized by any one of several forms of "loose" system coupling (such as disk coupling) between the Red and Black systems.

Protection of classified data processing and transmission needs little discussion because the feasibility is evident and has been already demonstrated by use. Prevention of classified data leakage is the major design feature, and must be properly implemented to ensure that leakage cannot occur. This topic is discussed in the following paragraphs.

Consider first the mailbox concept itself. At the present time it is widely used in computer systems for purposes varying from intersystem communication to interprocess communications in a multiprogrammed operating system. Because the mailbox (however implemented) is a shared resource, means must be devised to ensure that it is shared in an orderly fashion (to avoid errors, deadlock, lockout, etc.). For example, if the mailbox is a two port disk, the disk capacity must be suitably allocated for intersystem message storage. In addition, it must be absolutely secure. A typical secure RED-BLACK communications scenario might be as follows:

(Assumption: The shared disk features a smart microprocessor-based controller which independently manages the disk resource (allocates and deallocates space, etc.) and communicates with both the Red and Black Systems.)

	Communication Parties	Message	Action
1.	Black to Controller	Message for Red	---
2.	Controller to Black	Ready, to receive message	Black sends message in standard format to controller. Controller stores it on disk in a location unknown to Red.
3.	Controller to Red	Message from Black	---

4.	Red to Controller	Ready, to receive message	Controller retrieves message for Red from an area unknown to Red, and transmits it to Red. Red stores the message in a suitable area in memory.

6.5 Security Evaluation. The security evaluation of the proposed design is mainly concerned with the question, "Is there any way (consistent with the given assumptions) that any one of the designated potential agents can penetrate the Red system and gain access to classified plain text data?"

Given the assumptions in the system design, it seems clear that the system is secure against either accidental penetration or simple intentional penetration efforts. Note that the Red and Black systems are physically completely separate, with no communications except through the smart mailbox. If we delete the smart mailbox, then there are no communications and the complete security of the design is clearly assured. Thus, it is obvious that, if the Red system is to be penetrated at all, penetration must be via the smart mailbox. Therefore, we focus our security evaluation on that feature of the system.

Consider two basic questions which, taken together, cover all penetration possibilities. 1) Is there any way that the cryptographic key can be made to appear in the smart mailbox as a result of some Black message? 2) Is there any way that classified plain text from the Red Processor can be made to appear in the smart mailbox as a result of some Black message?

The answer to the first question is no, because the cryptographic key is not known to the Red Processor (i.e., it is not stored in the memory of the Red Processor). The cryptographic key is known only to the cryptographic units, and cannot be read by either the Red or Black Processors.

The answer to the second question is also no, provided that detail design of the Red system is suitable (i.e., error-free in design if the system is operating properly, and security fail-safe in design if there is a Red hardware malfunction). The detailed design would be quite simple, except for the requirement that each message from the Red Processor to the Black must contain some plain text information (such as information on intended recipients for the data from Red, or the destination address of the data from Red).

Given the above requirements, it is necessary to ensure that classified data can never be made to appear in the plain text portion of a Red-to-Black message as a result of any possible Black to Red message. This assurance must be provided by the Red Processor hardware and/or software. One possible scheme is to store all classified data for transfer into a fixed Red memory buffer area for transfer to the smart mailbox, and

alter the Red system hardware in such a way that any data transferred to the disk from this area is automatically encrypted by hardware (i.e., provide an encrypted-read-only Red memory buffer area). To complete this scheme, it must also be assured that the Red memory area serving as buffer area for plain text to the smart mailbox can never contain classified data. This can be done by using a data tag scheme in the Red Processor, such that all data is tagged to show whether it is classified or not (this is a special case of what is sometimes called "capability-based addressing"). The Red buffer area for transmitted plain text would then be unclassified write-only, a discipline enforced by hardware protect.

6.6 Performance/Cost Evaluation. Questions addressed in this section are:

- Would the proposed RED-BLACK scheme provide adequate performance and capacity for its intended applications?
- Would the cost be reasonable when compared with the costs of nonsecured systems of equivalent performance and capability?

The answer to the first question is difficult to provide in an absolute sense, since it is not known what the total performance and capacity requirements are for the target secure system. However, the proposed scheme is not restrictive in this regard, because a Red system of any desired performance and capacity (processor speed, memory size, secondary storage size, etc.) may be used.

The main total system performance *obstructions* are in the following areas:

System Component	Area of Concern
• Red Processor	Unclassified-write-only feature
• Cryptographic Unit	Encrypted-read-only feature
• Smart Mailbox	Mailbox size, speed of message transfer

Considering the obstructions in order of least concern first, we consider the Red Processor unclassified-write-only feature (used in the Red buffer for transfer to smart mailbox). Each piece of Red data to be written in this area must be associated with a tag indicating whether it is classified or not. The Red Processor unclassified-write-only feature prevents the writing to the buffer of any data which is classified by tag checking. The tag checking must be done with every piece of data, but it can be accomplished rapidly using standard hardware methods and should have little effect on the overall system performance.

The performance of the cryptographic unit used with the encrypted-read-only is likely to be more of a problem. For example, Motorola now markets an LSI-based cryptographic unit for the NBS standard encryption scheme which requires 160 microseconds for encryption/decryption of 8 bytes of data, or 20 microseconds for encryption/decryption of each byte of data.

For a similar device, Stanford (optimistically) estimates that the encryption job can be done in 1 microsecond by an LSI chip, while NBS is estimating 40 microseconds (or 5 microseconds per byte). Thus, it appears that the additional time required to encrypt/decrypt large data files with a suitable secure scheme may be fairly significant (for the near future anyhow).

The performance of the smart mailbox is the most critical potential obstruction in the proposed scheme. Limiting the performance of the smart mailbox are the processor-to-mailbox dialogs that must take place, and the technology used to implement the smart mailbox.

The dialogs constitute an overhead which is most troublesome if typical messages are short. However, it should be understood that in any multiprocessor system, any workable communication scheme using some form of shared storage must inevitably be associated with some overhead due to the need for management of the shared resource. For example, if the mailbox is built from shared main memory, devices such as Dijkstra's P and V operators and semaphores are needed to ensure that one processor is not writing into a shared area at the same time another processor is reading from it. Thus, some smart mailbox management overhead is inevitable regardless of the particular mailbox storage technology used.

Disk technology is a promising candidate for mailbox storage because it is quite fast and yet provides a large mailbox (for big messages or large numbers of messages) at a fairly low cost. Two ports may be provided for the disk to allow concurrent message transfers, and this potentially doubles performance over a single port scheme. For example, the Black Processor can be writing one message in the disk at the same time the Red Processor is reading a different message from it. However, if a disk mailbox is too slow, it is possible to significantly improve smart mailbox performance by building it with another storage technology, such as drum or a separate segment of main memory.

If disk technology is used, data transfer rates may be high (800K bytes per second) and performance will be fairly good (relative to a RAM mailbox) for longer messages. However, for short messages the large access average access time of a disk (typically around 25 ms) relative to core (typically about 1 μ s) may make the device up to 25,000 times slower than an equivalent RAM mailbox. Whether or not this fact would make the disk mailbox unacceptable for the human user (response times in seconds) has not been determined.

The RED-BLACK multiprocessor cost evaluation may be considered as follows:

- Consider the cost of multiprocessor system versus uniprocessor systems, generally.
- Consider the additional cost of special security features. The features are briefly discussed in the following paragraphs.

6.6.1 Multiprocessor Versus Uniprocessor System Cost. Whether or not a multiprocessor system is cost-effective relative to a uniprocessor system of equivalent performance is a subject which has been debated in the computer industry for many years. The question has not been fully resolved, and it may never be. Suffice it to say here that dual processor systems are now quite common, having survived the cost/effectiveness test of the marketplace. Therefore, we may conclude that, while all possible multiprocessing systems may not be cost-effective, some certainly are. We assume that the RED-BLACK Multiprocessor scheme proposed here can be suitably designed so as to fall in the latter class.

6.6.2 Special Security Features Cost. The special security features proposed are:

- Use of cryptographic units for Red processor encrypted-read-only.
- Use of data tags for Red processor unclassified-write-only.
- Use of a smart mailbox.

Cryptographic units can now be built with LSI technologies at low cost. For the NBS standard encryption scheme, Motorola now has a small board selling for about $500, and expects to offer a hybrid device soon for about $150. Fairchild offers an NBS standard unit built from four chips, with the price of the four chips currently about $30. Thus, we conclude that the cost of a cryptographic unit will not be very high, mainly because it can be built using low cost LSI technology.

The unclassified-write-only feature for the Red Processor is a special case of capability based addressing, a technique which is considered by some to be an attractive partial solution to the secure computer system design problem. To implement full-fledged capability-based addressing requires a new Red Processor architecture, and this can be extremely expensive to implement (assuming the Red Processor does not already possess the feature). For an interim solution, this feature would probably be best provided by a special box which intercepts all Red Processor/Memory transactions and imposes the desired control. Such a box could probably be designed and built on a one-time basis for a few thousand dollars.

The major hardware cost item for the proposed design would be the smart mailbox. At present, it could best be built using microprocessor-based disk controller (assuming the disk mailbox offers sufficient performance). A suitable disk (14 megabyte capacity) costs about $10,000 and the controller could be designed and built by Harris ESD for about $30,000. However, it is possible that a suitable smart disk control unit already exists on the market which could be modified for the job (there is a general trend toward the use of smart disk controllers). If so, substantial savings might result by purchasing and modifying such a device.

6.7 Summary and Conclusions. A RED-BLACK Multiprocessing scheme has been described as a promising solution to the Secure Data Base System design problem. The scheme is conceptually simple, demonstrably secure,

offers good performance, and can be implemented from conventional computer system components with relatively small additional cost for the security features.

A typical system using this scheme is basically a loosely-coupled two processor multiprocessing system which employs cryptography for classified data transmission and a smart mailbox for communications between the two processors. The smart mailbox can be constructed using disk storage and a microprocessor-based controller; but for higher performance, it can be built using a drum or RAM for message storage. Cryptography and the smart mailbox (with associated Red Processor memory address modifications) are the keys to the system's security. Security is easily demonstrated, mainly because of the conceptual simplicity of the scheme employed. The basic system cost should be only slightly higher than the cost of a similar multiprocessing system without the additional security features, because the latter can be provided for an estimated $40,000 or less.

Overall, it is concluded that the proposed design approach is a promising one, because it is demonstrably secure while providing potentially high performance, capacity and cost effectiveness.

SECTION 7.0

DATA BASE GUARD APPROACH

7.0 DATA BASE GUARD APPROACH

This section examines the feasibility of a multiported disc controller that acts as a guard between the data base and one of several computers. Control is accomplished by allowing access to only the secret computer. Attempts to access a Top Secret record will be disallowed by guard checks of the record classification.

7.1 A Protection Scheme. When a data base resides in a disk storage system and the users are separate computer systems it is possible to protect sensitive data from unauthorized access by building the protection mechanization into the multiported disk controller. Consider the system architecture shown in Figure 40 and the following protection mechanism. A multiported disk system is connected to several user computer systems with each storage system interface assigned an access privilege level. Every sector on the disk contains a security access word as part of the sector format. When a user computer requests an access to a disk sector the controller reads the access control portion of the disk format first and only honors the request if the sector has a security level less than or equal to the access privilege level assigned to the users I/O interface. This scheme provides a multiported storage system with hardware protection of sensitive data. Several questions which must be answered in order to evaluate the data protection scheme are:

- Does this scheme adequately protect the data records in various file structures?
- What constraints does the protect scheme place on the selection of file structure or on the logical records to physical-storage mapping?
- How can write protection be implemented; i.e., how does the data base change?
- How can space allocation be handled?
- How is the access speed of the system affected by the protection scheme?

The answers to these questions must be found by examining some of the mechanics of file structures and systems.

7.2 The Data Base: Files and Records. The data stored in a data base is contained in logical entities called records and is the smallest amount of information that can be directly accessed. The records are grouped together into larger structures called files. Physical storage of a file is usually made up of a number of blocks which are the smallest physical access that a file system makes to the storage device. The logical records of a file are mapped onto the blocks. For disk storage systems this is usually an integer number of disk sectors. The blocks of a file are identified by the "file access mechanism." There are hundreds of file access mechanisms or structures but in the interest of brevity only two of the most common will be examined. One is called block linking and the other is record indexing. Figure 41 illustrates both methods. Linked files are often used for files which are sequentially accessed. The access for one

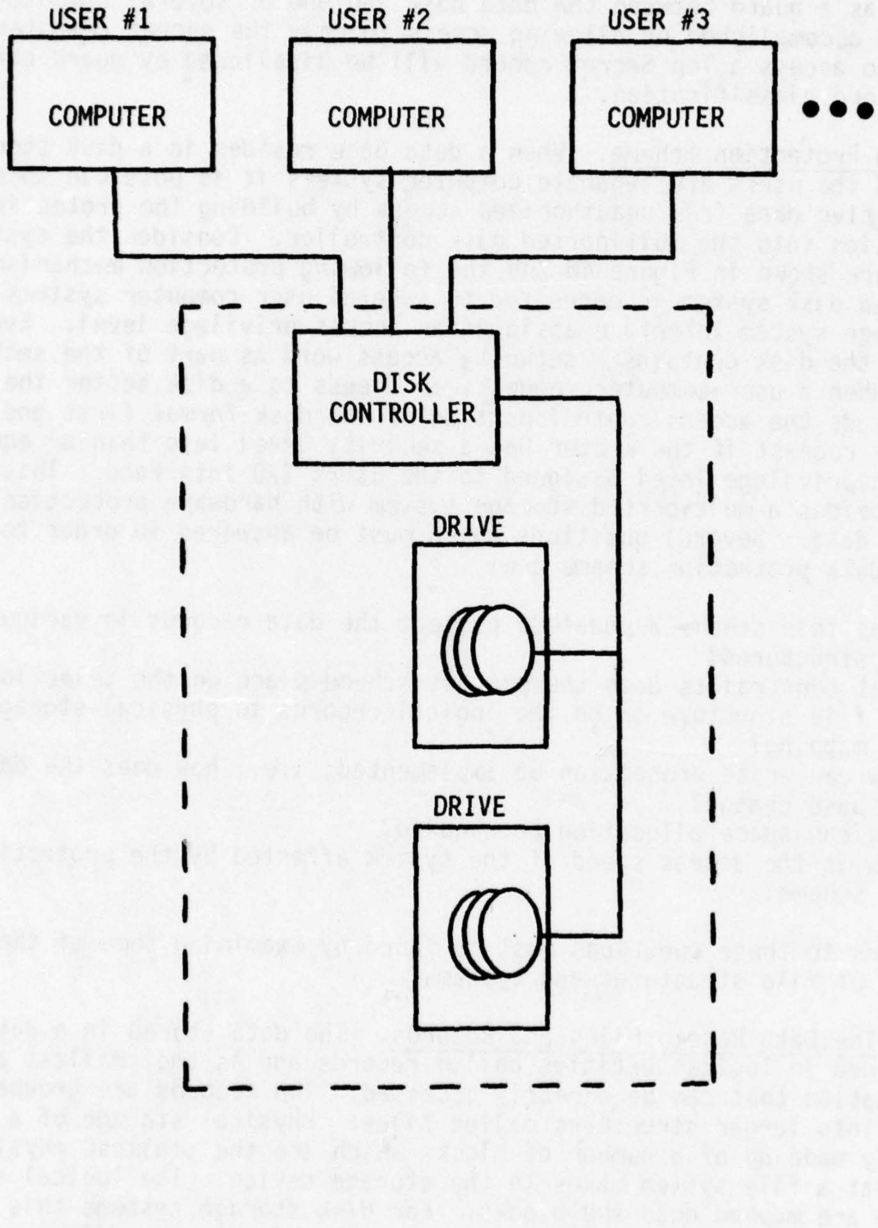
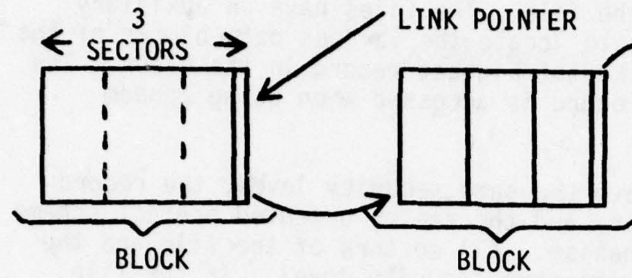
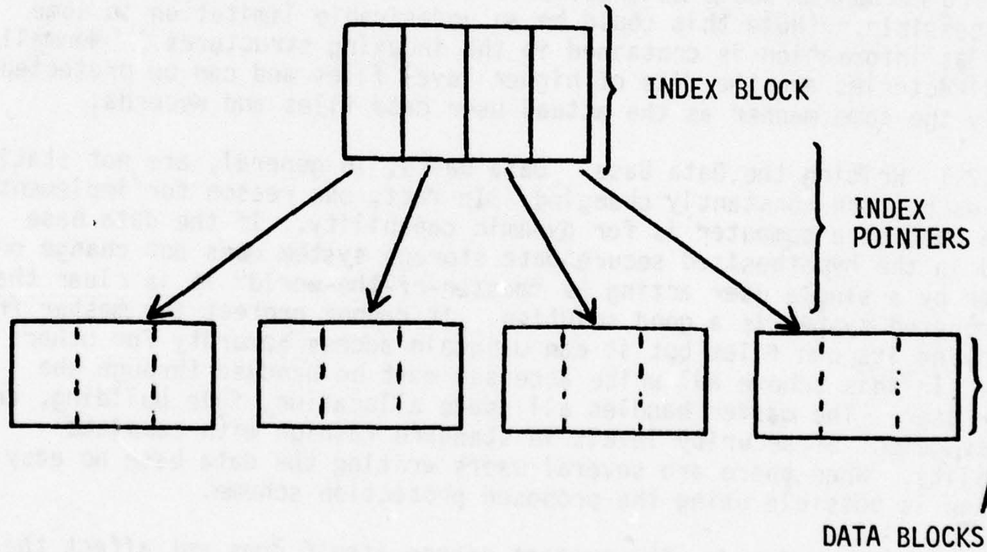


Figure 40 THE SYSTEM



FILE CONSISTS OF 3 BLOCK OF 3 SECTORS/BLOCK



FILE CONSISTS OF 4 DATA BLOCKS OF 3 SECTOR/BLOCK
AND 1 INDEX BLOCK CONTAINING ONLY 4 POINTERS

Figure 41

block of data gets the pointer for the next. Thus, there is very little overhead in either time or storage. Indexed files are used for random access files and have an access mechanism which allows more efficient random access to the records of the file. The files have an auxiliary structure which contains pointers to locate the various data blocks of the file. They may also contain the key of highest record in the block. The key is the "address" by which a record is accessed when using random accessing.

If the records in the file have the same security level, the record-to-storage mapping can be arbitrary and the sector oriented protect scheme will work with either access mechanism. All sectors of the file and the index structure if used can have the same security level. If the file contains records of different security classifications a linked structure is impossible since a lower classified user would not be able to follow the links. If the file is of the index type the only requirement is to ensure that two records of different security classifications do not share storage in the same sector. In general, it would also be necessary that the index blocks have the security level of the lowest record in the file or the accessing mechanism would be unaccessible to a user even though the data was accessible. (Note this could be an undesirable limitation in some cases, as information is contained in the indexing structures.) Normally file directories are the data of higher level files and can be protected in exactly the same manner as the actual user data files and records.

7.2.1 Writing the Data Base. Data bases, in general, are not static entities but are constantly changing. In fact, one reason for implementing a data base on a computer is for dynamic capability. If the data base stored in the hypothesized secure data storage system does not change or is written by a single user acting as "master-of-the-world" it is clear that the proposed system is a good solution. It cannot protect the master from destroying its own files but it can maintain access security for other users. In this scheme all write accesses must be handled through the master user. The master handles all space allocation, file building, and the assignment of security levels in standard fashion with complete visibility. When there are several users writing the data base no easy solution is possible using the proposed protection scheme.

7.2.2 Access Speed. The protect scheme itself does not affect the access speed of the proposed system architecture. Rather, it is the architecture which suffers access speed limitations. A sequential access controller can be used when disk access rates are slow, but when the number of users and the access rate increase, the architecture will suffer from a significant amount of disk arm movement and rotational latency. In this application, a sophisticated controller can improve system performance by using sector queuing and overlapping seeks with data transfer operations. This sophistication makes the controller look more like a complete processor based file system, than a simple controller.

7.2.3 Design. The following design is an example of a multiported disk controller which provides sector data protection. The design is a controller for the Diablo series 400 disc drives. User interfaces are intended to connect to PDP-11 users. The goal was to provide a basic design which could handle eight disk drives or a maximum storage of 170 million words (using the biggest Diablo 400 series drive) and service at least three users with easy expansion. Automatic handling of multiple seek operations complicates the controller but makes the design more realistic for median loading of the storage system.

Figure 42 shows the disk storage system design consisting of from one to eight Diablo disk drivers. A control bus runs from the controller to each drive in a wired or bus fashion. A bidirectional data cable and a clock cable run individually from each drive to the controller. Control and status information is passed from the controller to the drives via the common control bus while the data and clock signals are handled on separate cables. The PDP-11 computers are connected to the disk controller via the Controller Interface Units (CIU's) and the CIU Bus. Figure 42 shows the CIU Bus connecting one CIU to the next, however, the design does not preclude a multilegged star type configuration.

Figure 43 shows the sector format which will be supported by the disk controller. The format contains an error coded security tag as the first part of the sector. · The control will read the tag and pass the data portion of the sector to the requesting user if the user has a security level greater than or equal to the tag. The format is compatible with Diablo drive requirements.

7.2.4 The CIU Bus. The CIU bus is a simple communication bus between the disk controller and the individual CIU. Data transfers via the bus are 16 bits wide with a maximum transfer speed of about 700K words per second. Bus mastership is always held by the controllers. The bidirectional bus data lines (reference Figure 44) are time shared for data and address. The address is shown in the select format.

7.3 The CIU Software Interface. Figure 45 shows the software interface to the protected storage system. The interface consists of 4 registers in the device space of the Unibus. The first register specifies the drive and cylinder number. The second register specifies the track and sector number for the access and the length of the transfer in integer number of sectors. (Each sector contains 256 words of data.) The complete Unibus Transfer address is specified in the remainder of the second register and all of the third. The fourth register is the status and control register with a format very similar to the standard DEC interface.

The controller will support read operations for all CIU's. A read will only get the data portion of a disk sector. The write function is limited to special users as specified by a ROM table in the controller and will write the access level word and 256 words of data. The access word is ammended to the front of the data buffer to be written, i.e., for an N sector write the buffer would have to be NX257 words.

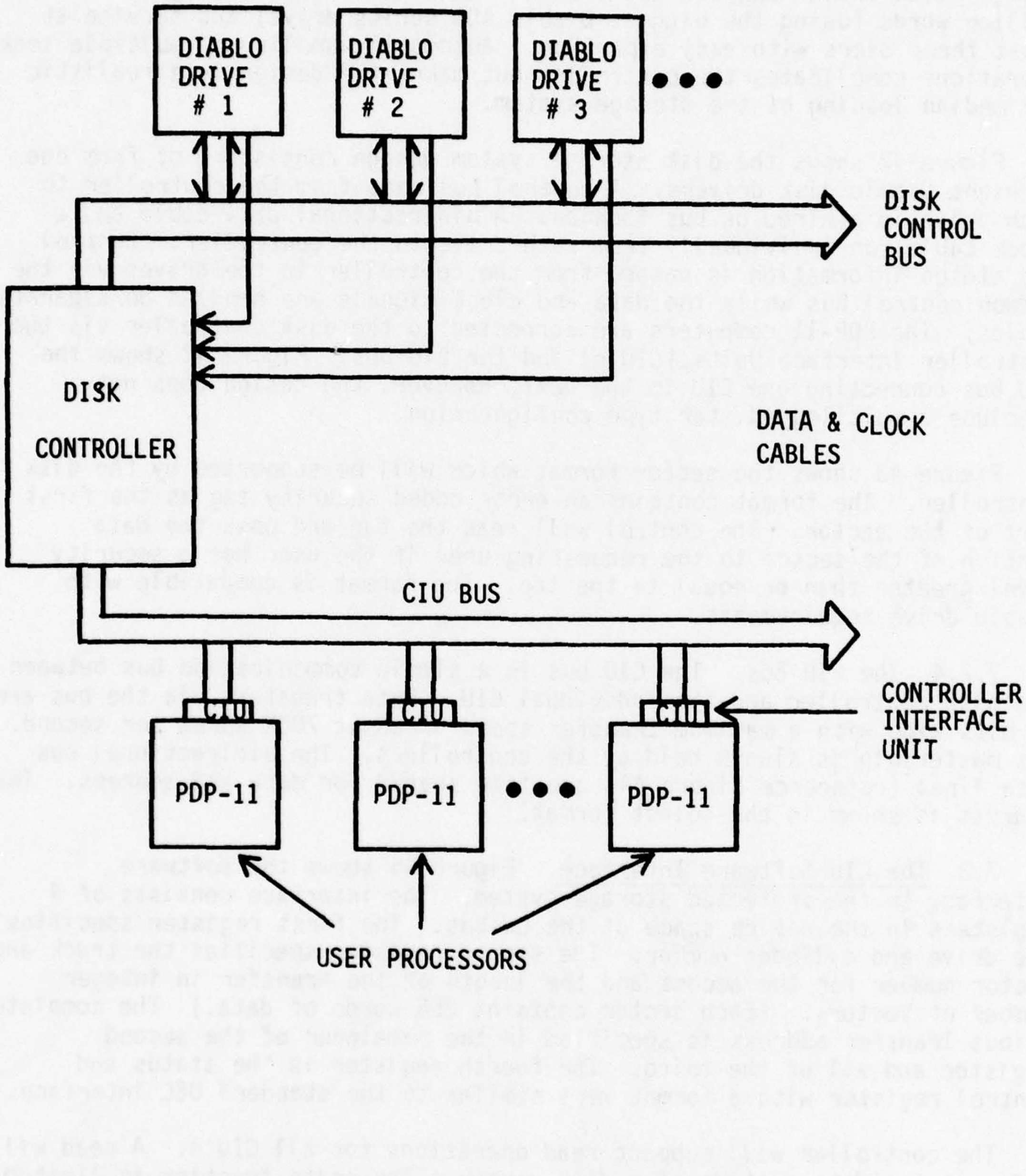
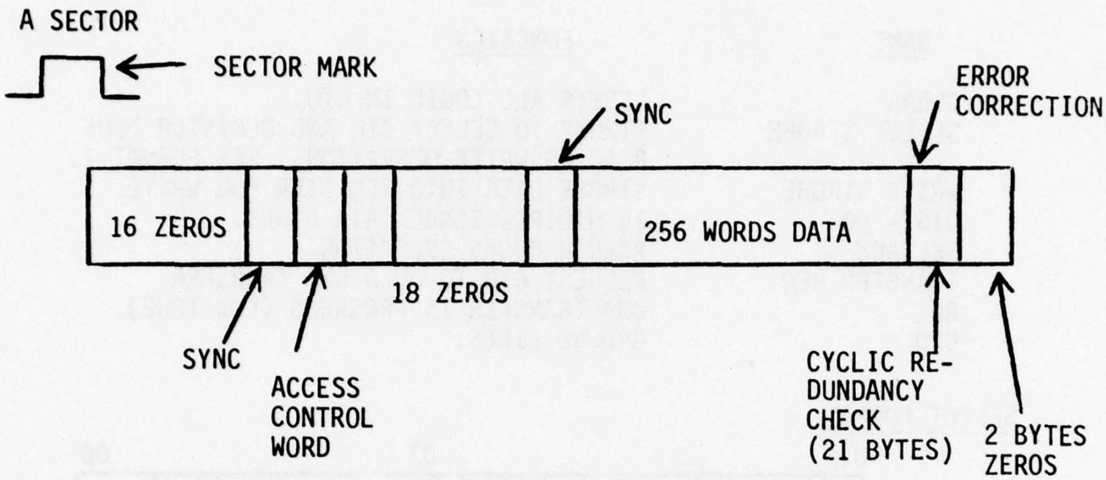

DIABLO DRIVE # 1
DIABLO DRIVE # 2
DIABLO DRIVE # 3
DISK CONTROL BUS
DISK CONTROLLER
DATA & CLOCK CABLES
CIU BUS
CIU
CIU
CIU
CONTROLLER INTERFACE UNIT
PDP-11
PDP-11
PDP-11
USER PROCESSORS

Figure 42

SECTOR FORMAT



SYNCS: FRAME SYNCHRONIZERS (8 DATA BITS LONG)
ACCESS CONTROL WORD IS ERROR DETECTION/ENCODE

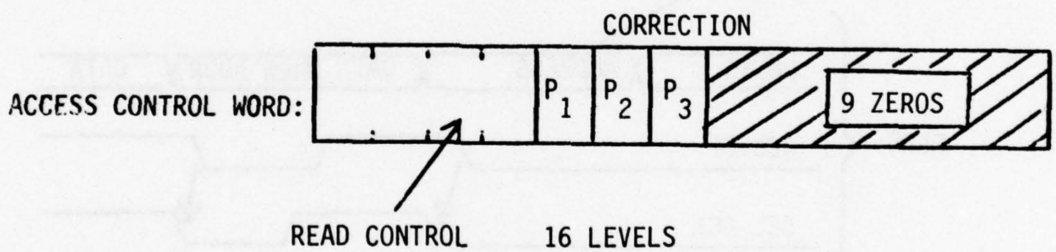
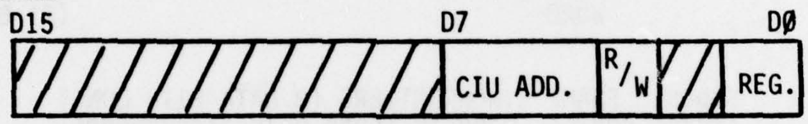


Figure 43 SECTOR FORMAT

THE CIU BUS

NAME	FUNCTION
CLEAR	RESETS ALL LOGIC IN CIU.
SELECT STROBE	STROBE TO SELECT CIU AND REGISTER PLUS READ OR WRITE OPERATION. SEE FORMAT 1.
WRITE STROBE	STROBE DATA INTO REGISTER FOR WRITE.
D15 - DØ	16 BIDIRECTIONAL DATA LINES.
SEL ACK	ACKNOWLEDGES SELECTION.
TRANSFER REQ.	REQUEST CIU TO DO A DMA TRANSFER.
RDY	DMA TRANSFER IS PROGRESS (LOW TRUE).
GND	GROUND LINES.

SELECT FORMAT



TIMING

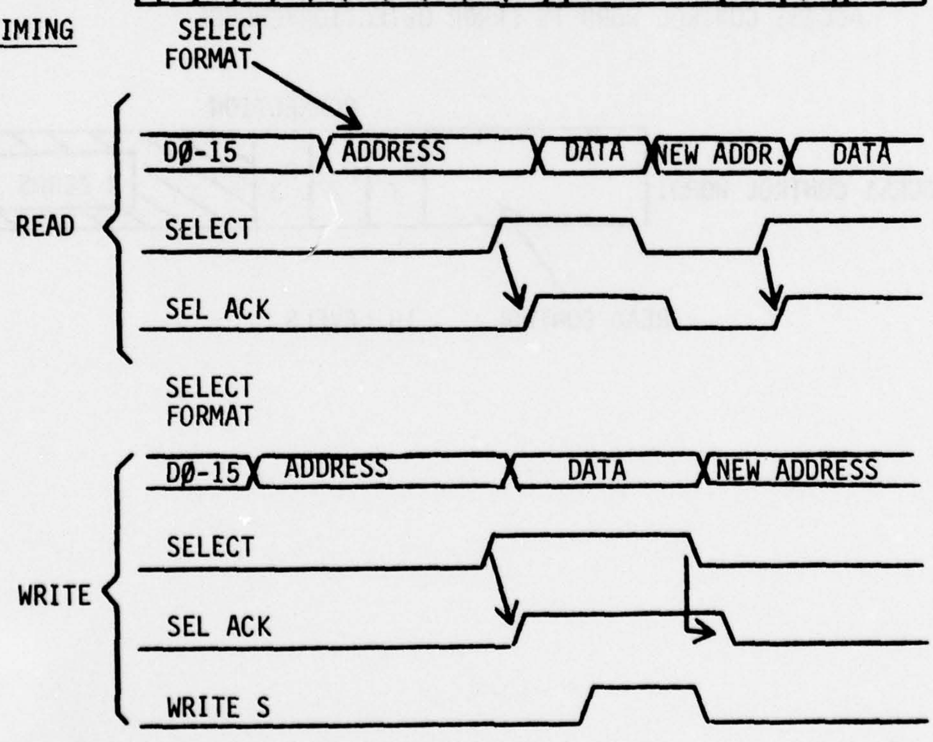


Figure 44 THE CIU BUS

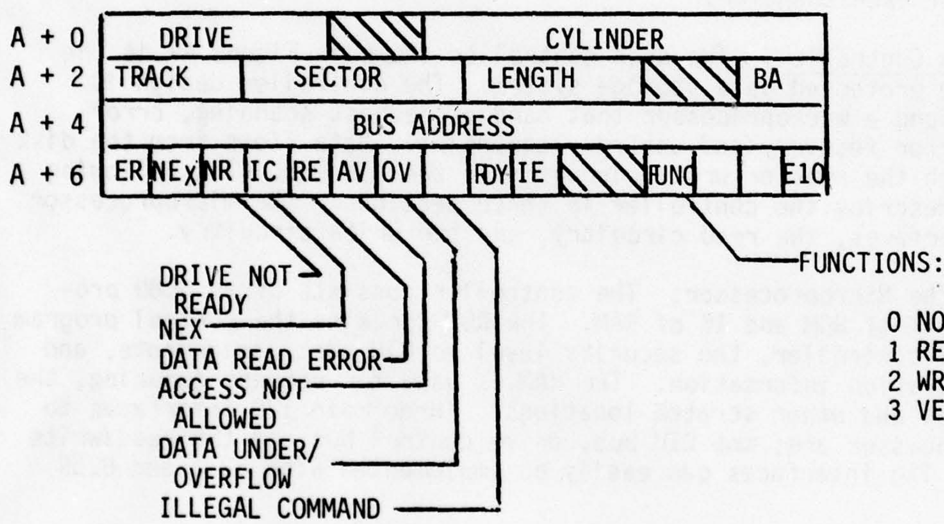


Figure 45 USER SOFTWARE INTERFACE

7.4 The CIU Hardware. The CIU is the hardware interface between the disk storage system and the user's PDP-11 processor. Figure 46 shows the basic block diagram of the CIU. The Unibus side of the CIU contains the basic address decoding and slave response logic to support four Unibus address locations as the I/O control registers. In addition, it contains logic to perform 16-bit DMA transfers. Because the registers of the CIU are accessed from two sources, the interface operates in a special manner. Prior to issuing a Go command the registers can not be accessed by the controller over the CIU bus, but can be read or written by the PDP-11 processor. Once the Go command has been issued the registers are released for access over the CIU bus and can not be meaningfully accessed from the Unibus. An access from the Unibus will always get zero. When the service request is finished the register will become accessable to the Unibus again.

DMA transfers between buses are accomplished when the controller has bus control of the interface. The address for the DMA transfer is contained in the address register of the CIU and is incremented by two at the completion of each transfer.

7.5 The Controller. The disk controller shown in Figure 47 is the heart of the protected data storage system. The controller design is centered around a microprocessor that handles request scanning, error checking, error recovery and control sequencing. Data flows from the disk drive through the read or write circuitry to the CIU bus. The following paragraphs describe the controller in three sections: the microprocessor and its interfaces, the read circuitry, and the write circuitry.

7.5.1 The Microprocessor. The controller consists of an 8080 processor with 4K of ROM and 1K of RAM. The ROM contains the control program for the disk controller, the security level to CIU port assignments, and some configuration information. The RAM is used for request queueing, the program stack and other scratch locations. Three main I/O interfaces to the microprocessor are; the CIU bus, drive control bus and the read/write circuitry. The interfaces can easily be implemented with 8212 and 8255 chips.

The drive control bus is the path the controller uses to command the drivers and is oriented for 8-bit microprocessors. Figures 48, 49 and 50 contain specifications of the bus as per Diablo documentation.

The microprocessor's function is to gather access requests from the users, send the appropriate commands to the drivers and the read/write circuitry, and monitor status and possible error conditions. Figure 51 is a simplified flow chart of the firmware that the microprocessor would execute. The complete firmware would include fault diagnostics and error recovery code along with initialization and setup operations. Sizing of the code shows that all the tasks can be accomplished in less that 3000 bytes of program and tables. The small amount of code makes verification of it easier and since it is in ROM it is safe from tampering.

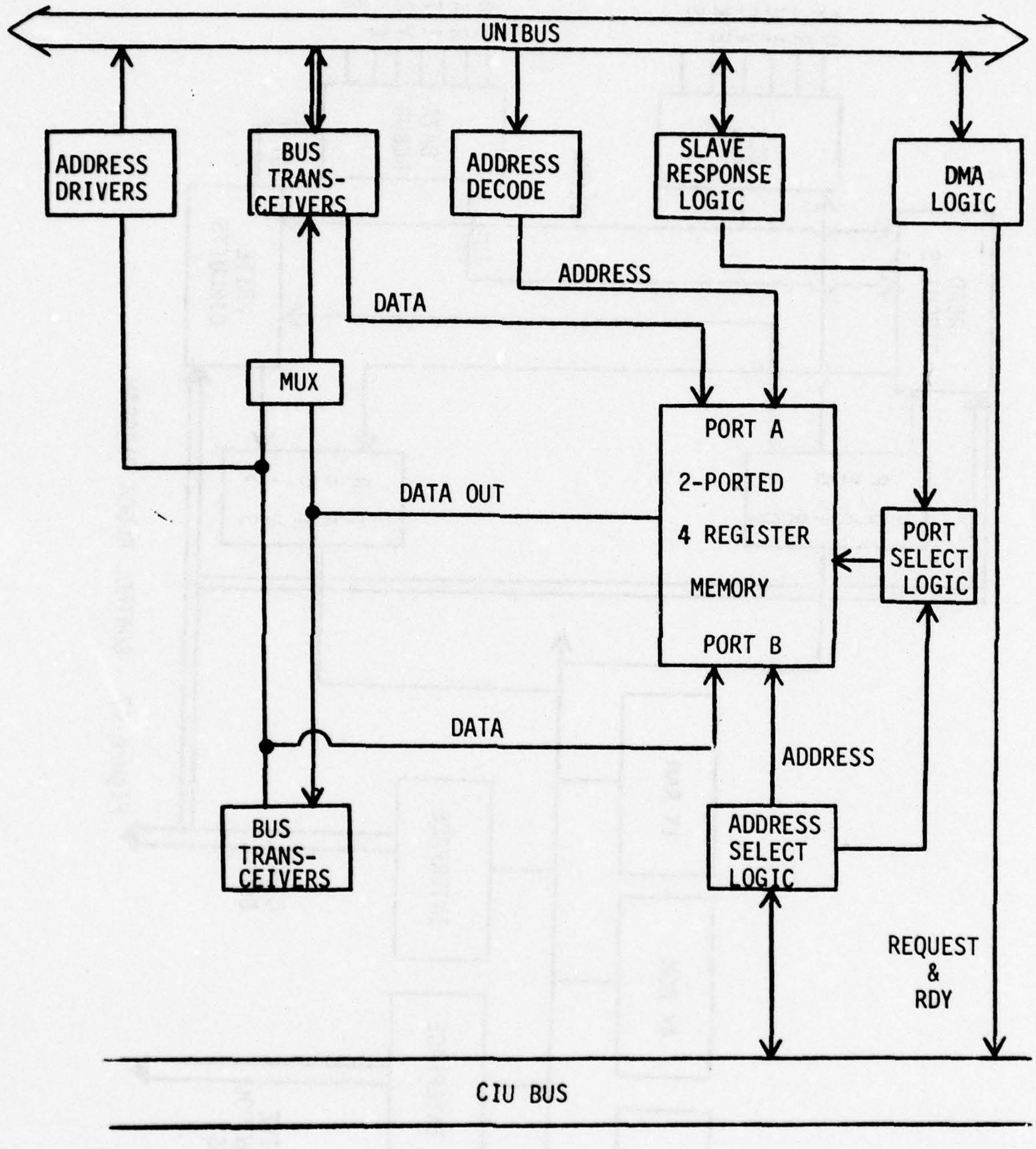


Figure 46 CIU HARDWARE

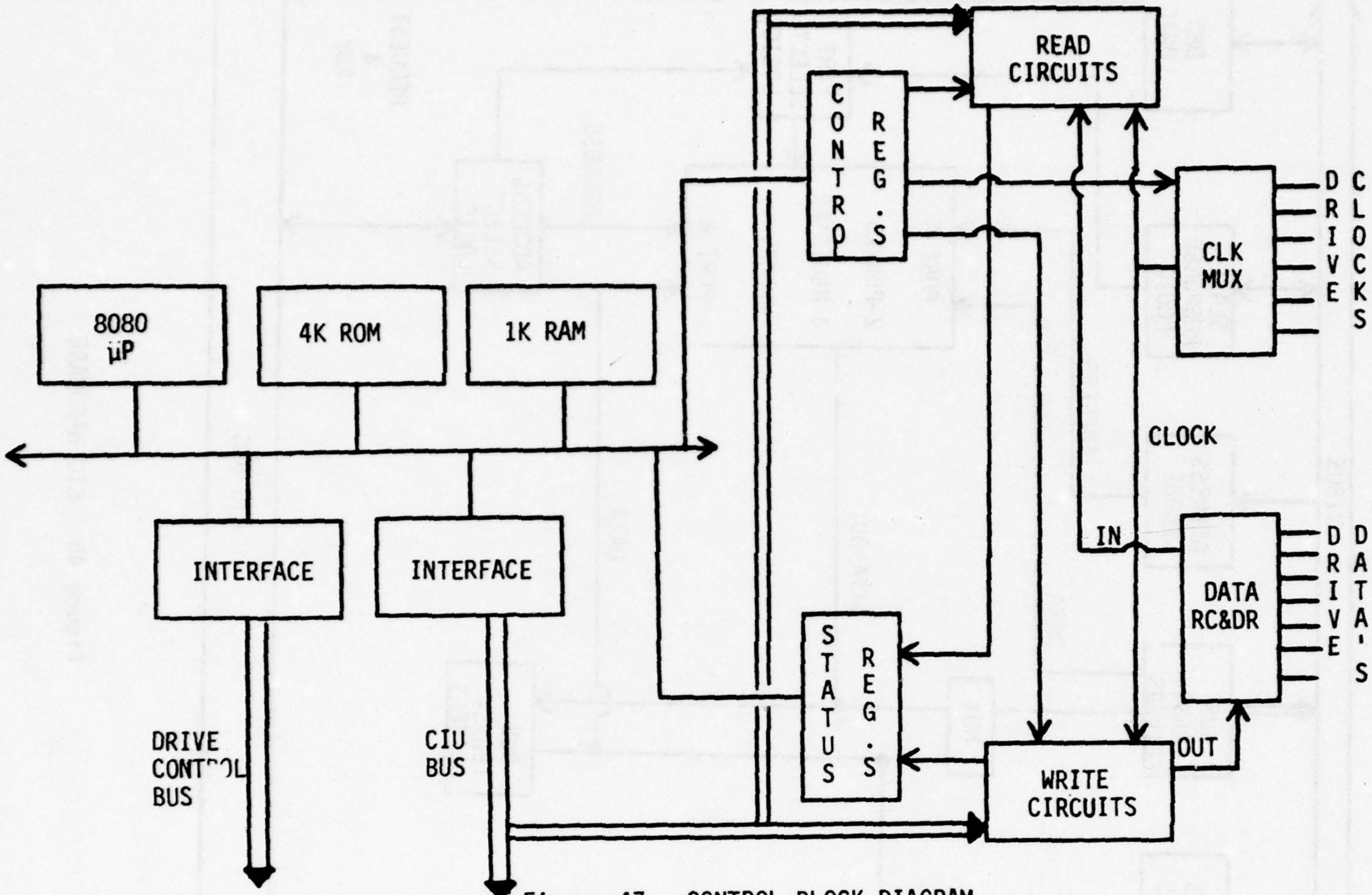


Figure 47 CONTROL BLOCK DIAGRAM

CONNECTOR 'A' PIN ASSIGNMENTS

PIN	SIGNAL	DESCRIPTION
1	GND 1	Ground.
2	Spare 1	Not used.
3-10	Control Bus (Bits 0-7)	Control Bus transfers disk unit select codes, command codes, and information bytes from controller to disk drive. Transfers status, mode, and sector counter bytes from disk drive to controller.
11	-SYSTEM CLEAR	When low, this signal deselects disk drive and clears input and output ports.
12	+SELECT STROBE	Strobes disk unit select code from control bus into drive.
13	+INPUT STROBE A	Strobes restore, cylinder and head commands into the input port. Also, sets "Input Full" and "Busy" status in mode byte.
14	+INPUT STROBE B	Strobes all commands (other than restore, cylinder and head) into the input port. Also, sets "Input Full" status in mode byte.
15	+OUTPUT ENABLE	Gates the contents of the disk drive output port onto the control bus, to be read by the controller. Also, resets "Output Full" mode bit.
16	+MODE ENABLE	When high, this signal gates the mode byte of the selected unit onto the control bus.
17	+SECTOR ENABLE	When high, this signal gates the contents of the sector counter for the selected unit onto the control bus.
18	+ATTENTION PULSE	A 500 ns pulse transmitted anytime an important change of status occurs in any unit.
19	+SECTOR PULSE	Indicates the beginning of each sector for timing the start of read or write. Also represents period of stable sector count.

Figure 48. Drive Control Bus Specifications (1 of 2)

CONNECTOR 'A' PIN ASSIGNMENTS (Continued)

PIN	SIGNAL	DESCRIPTION
20	+WRITE GATE	Controls beginning and end of data writing. +WRITE GATE should remain high for two bit periods after last data bit intended to be written on disk.
21	+READ GATE	Controls beginning and end of data reading.
22-24	Spare 2-4	Spare pins.
25	Spare 5	Not used.
26	GND 2	Ground.

Figure 48. Drive Control Bus Specifications (2 of 2)

COMMAND FORMAT

COMMANDS	CLASS	CONTROL BUS 7	6	5	4	3	2	1	0
Restore		0	0	0	0	0	0	0	1
Cylinder (followed by 2 bytes)	A	0	0	0	0	0	0	1	0
Head (followed by 1 byte)		0	0	0	0	0	0	1	1
Status Request A		0	0	0	0	0	1	0	0
Status Request B		0	0	0	1	0	1	0	0
Status Request C	B	0	0	1	0	0	1	0	0
Status Request D		0	0	1	1	0	1	0	0
Reset Write Protect*		0	0	0	0	0	1	0	1
Set Write Protect*		0	0	0	1	0	1	0	1
Track Offset In +3△*		0	0	1	1	0	1	1	0
Track Offset In +2△*		0	0	1	0	0	1	1	0
Track Offset In +△*		0	0	0	1	0	1	1	0
On-Track*		0	0	0	0	0	1	1	0
Track Offset Out -△*		1	0	0	1	0	1	1	0
Track Offset Out -2△*		1	0	1	0	0	1	1	0
Track Offset Out -3△*		1	0	1	1	0	1	1	0
Diagnostic Test A*		0	0	0	0	0	1	1	1
Diagnostic Test B*		0	0	0	0	1	0	0	0
Diagnostic Test C*		0	0	0	0	1	0	0	1

*Not to be issued when drive is busy.
△Represents fraction of track width; typically 100 microinches.

Figure 48. Drive Control Bus Specifications (1 of 2)

ADDRESS FORMAT

ADDRESS	CLASS	CONTROL BUS 7	6	5	4	3	2	1	0
Unit Select Code	C	X	X	X	U_{16}	U_8	U_4	U_2	U_1
Cylinder Address MS (2nd byte)		X	X	X	X	X	X	C_{512}	C_{256}
Cylinder Address LS (1st byte)	B	C_{128}	C_{64}	C_{32}	C_{16}	C_8	C_4	C_2	C_1
Head Address		X	X	X	X	H_8	H_4	H_2	H_1

NOTE: Class A commands use Input Strobe A, Class B commands use Input Strobe B, and Class C commands use Select Strobe.

Figure 48. Drive Control Bus Specifications (2 of 2)

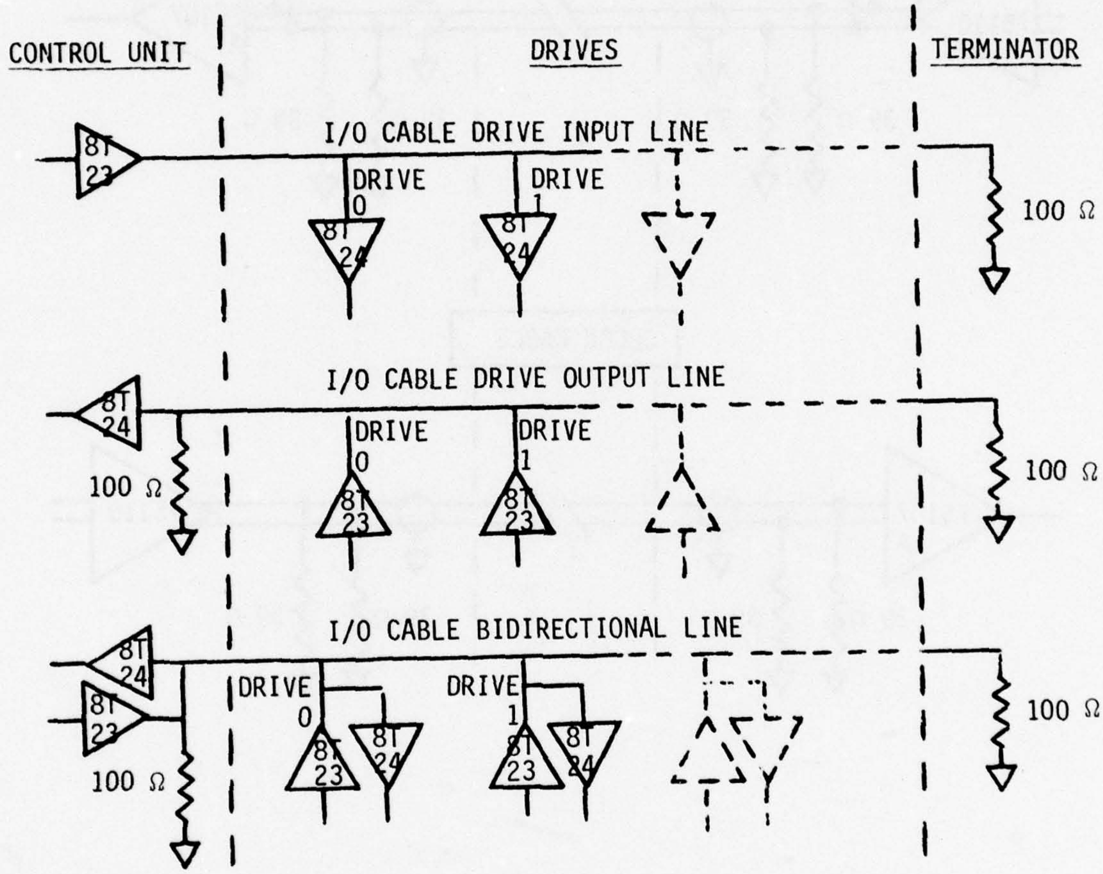


Figure 49 I/O CABLE TERMINATION

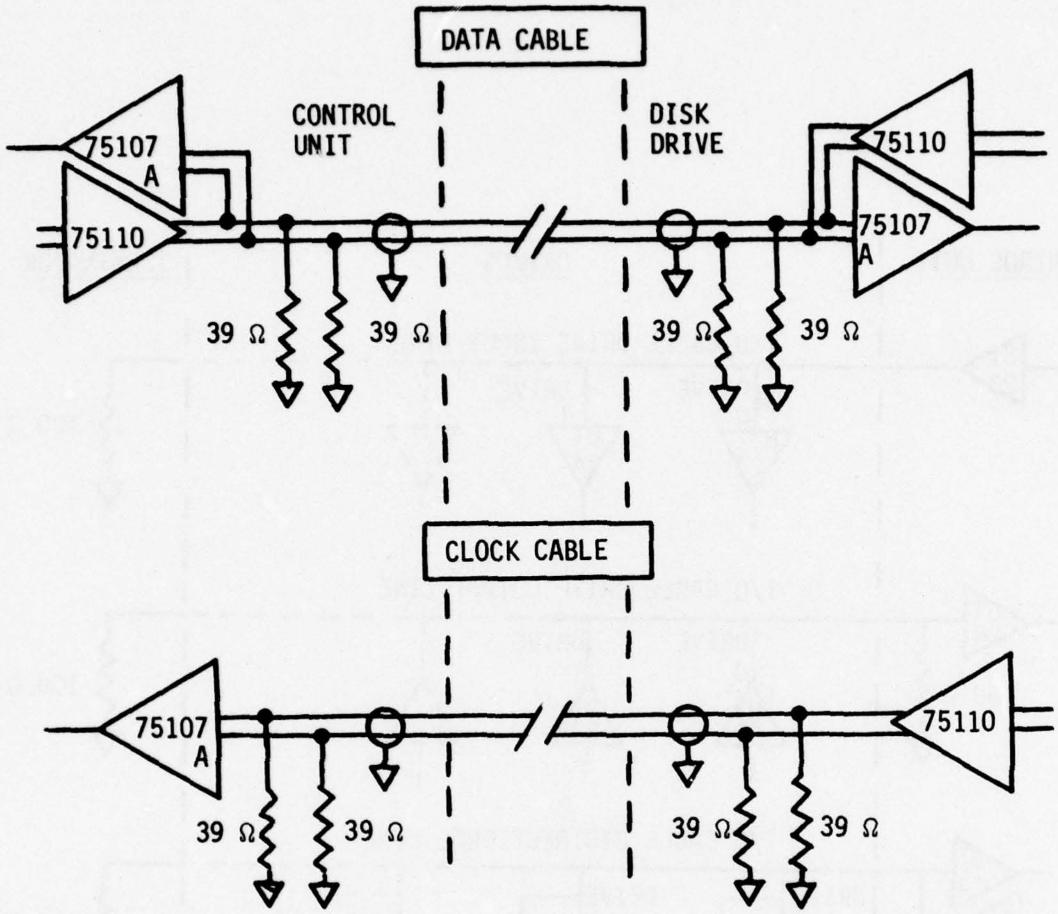


Figure 50 DATA & CLOCK CABLE TERMINATION

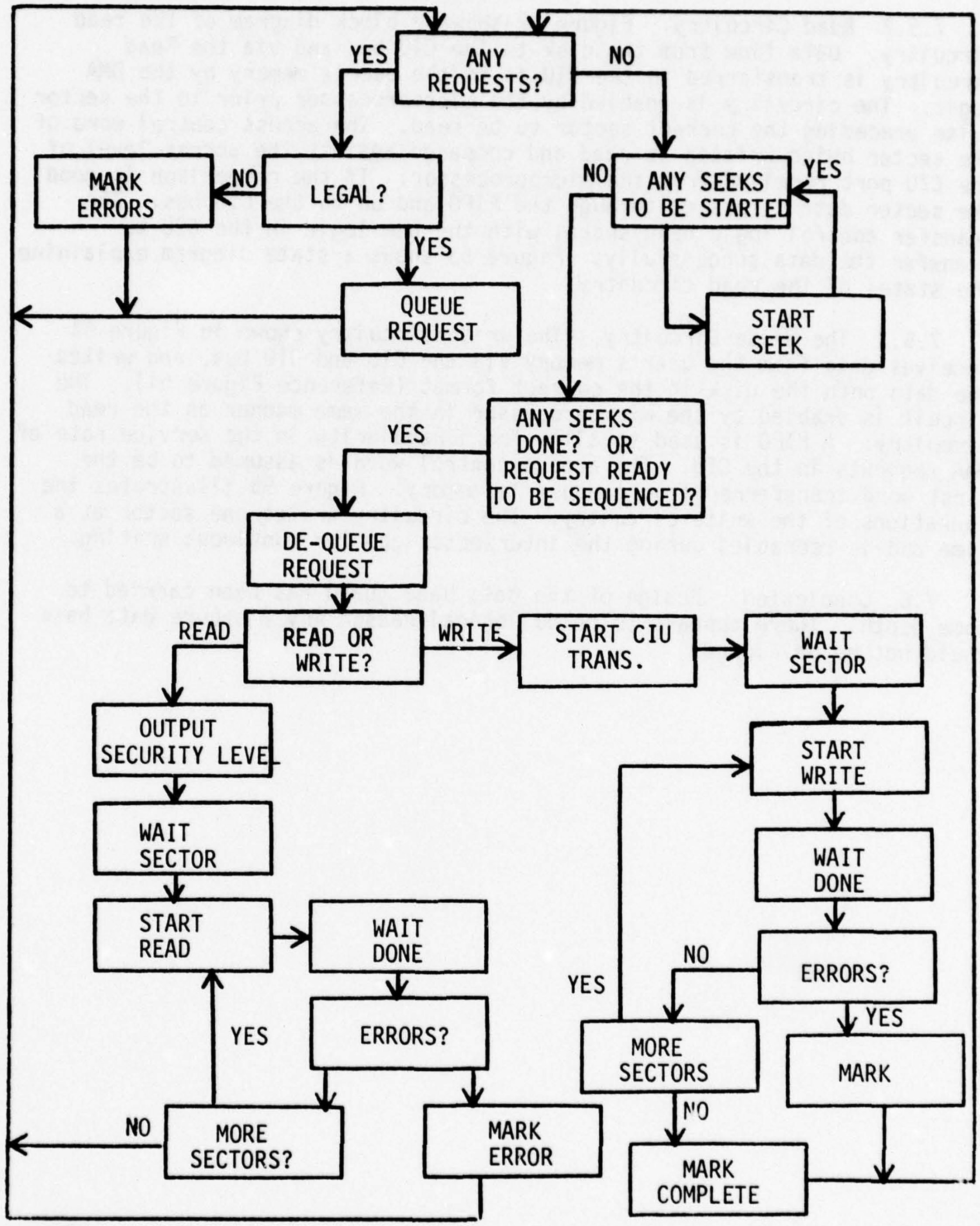


Figure 51 CONTROLLER FLOWCHART

7.5.2 Read Circuitry. Figure 52 shows a block diagram of the read circuitry. Data flow from the disk to the CIU bus and via the Read Circuitry is transferred to the CIU in to the user's memory by the DMA logic. The circuitry is enabled by the microprocessor prior to the sector pulse preceding the correct sector to be read. The access control word of the sector being written is read and compared against the access level of the CIU port received from the microprocessor. If the comparison is good the sector data is passed through the FIFO and on to the CIU bus. The transfer control logic hand shakes with the DMA logic in the CIU to transfer the data successfully. Figure 53 shows a state diagram explaining the states of the read circuitry.

7.5.3 The Write Circuitry. The write circuitry shown in Figure 54 receives data from the user's memory via the CIU and CIU bus, and writes the data onto the disk in the correct format (Reference Figure 54). The circuit is enabled by the microprocessor in the same manner as the read circuitry. A FIFO is used to allow for irregularity in the service rate of DMA requests in the CIU. The access control word is assumed to be the first word transferred from the user's memory. Figure 55 illustrates the operations of the write circuitry. The circuitry writes one sector at a time and is reenabled during the intersector gap for continous writing.

7.6 Conclusion. Design of the data base guard has been carried to some depth. There appears to be no logical reason why a secure data base could not be developed.

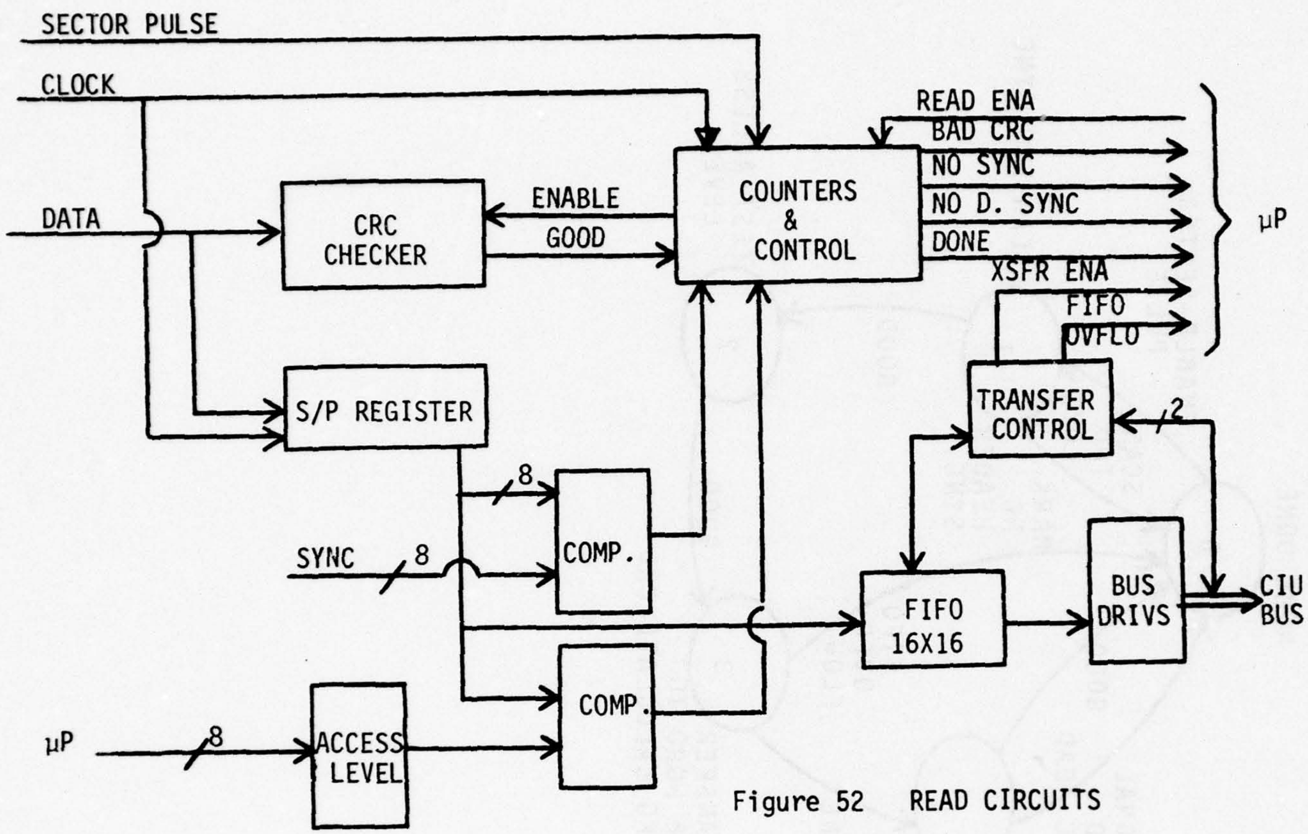


Figure 52 READ CIRCUITS

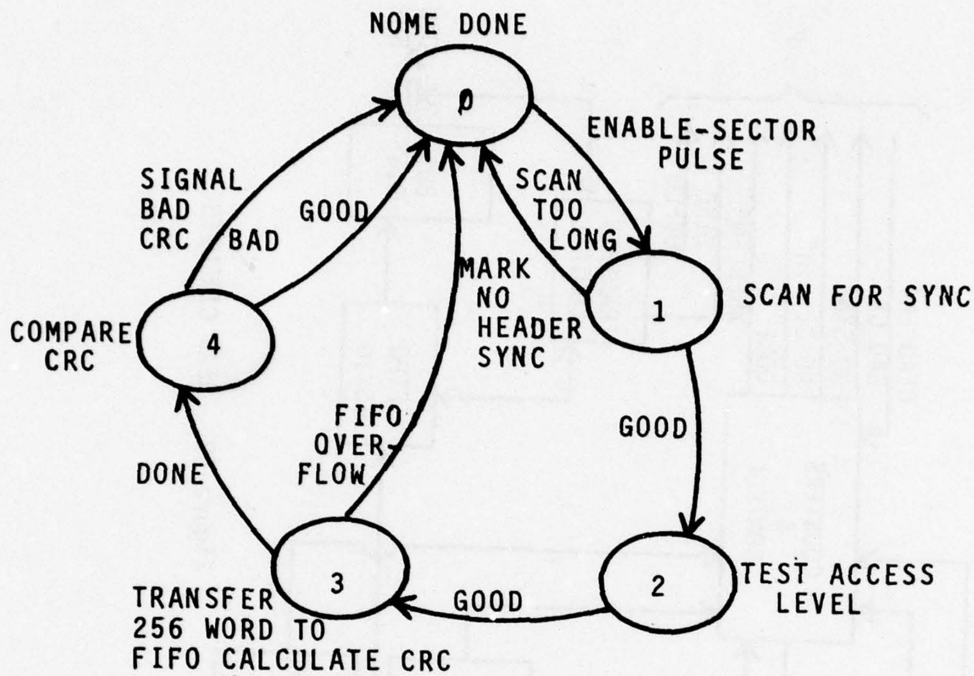


Figure 53 READ STATES

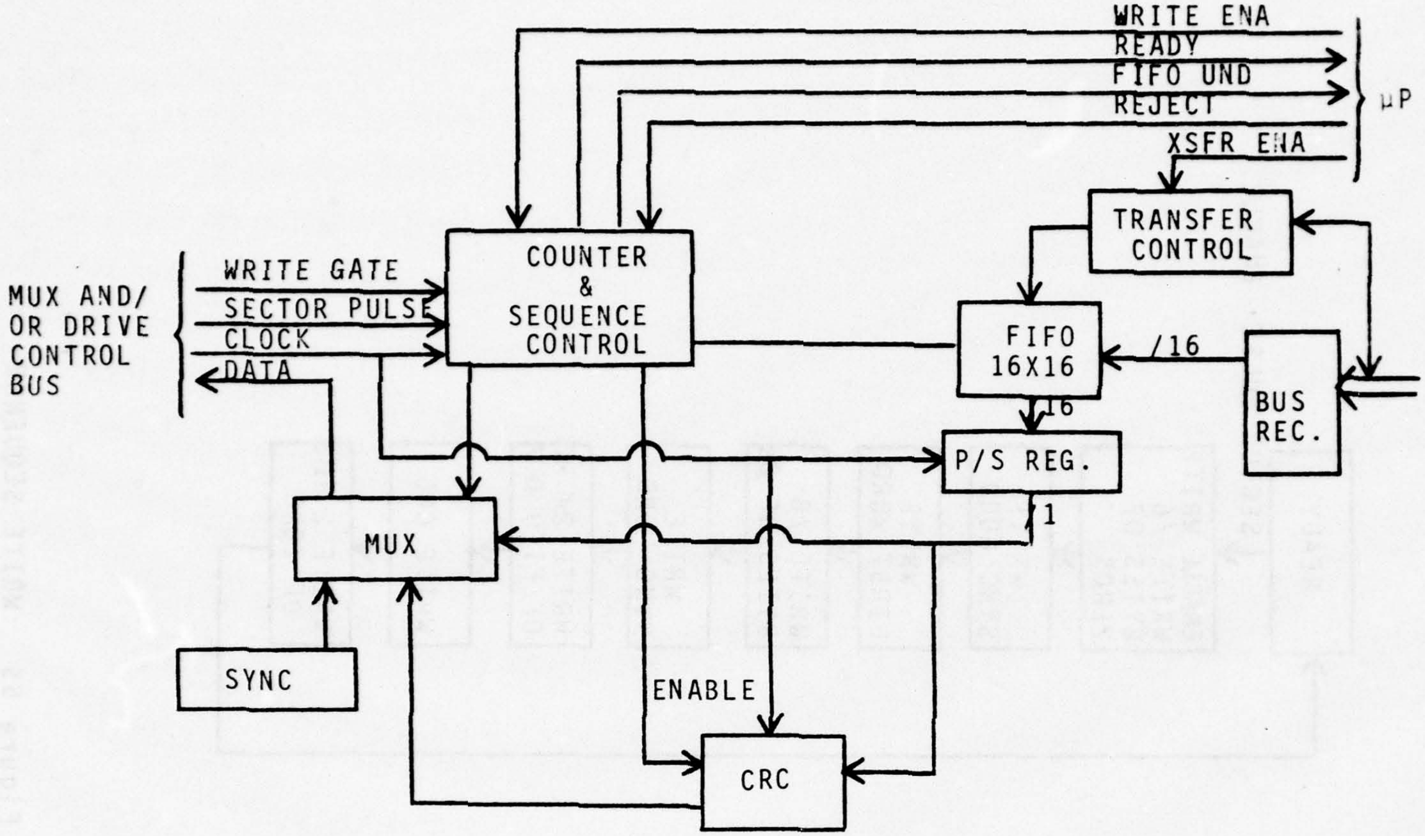


Figure 54 WRITE CIRCUIT

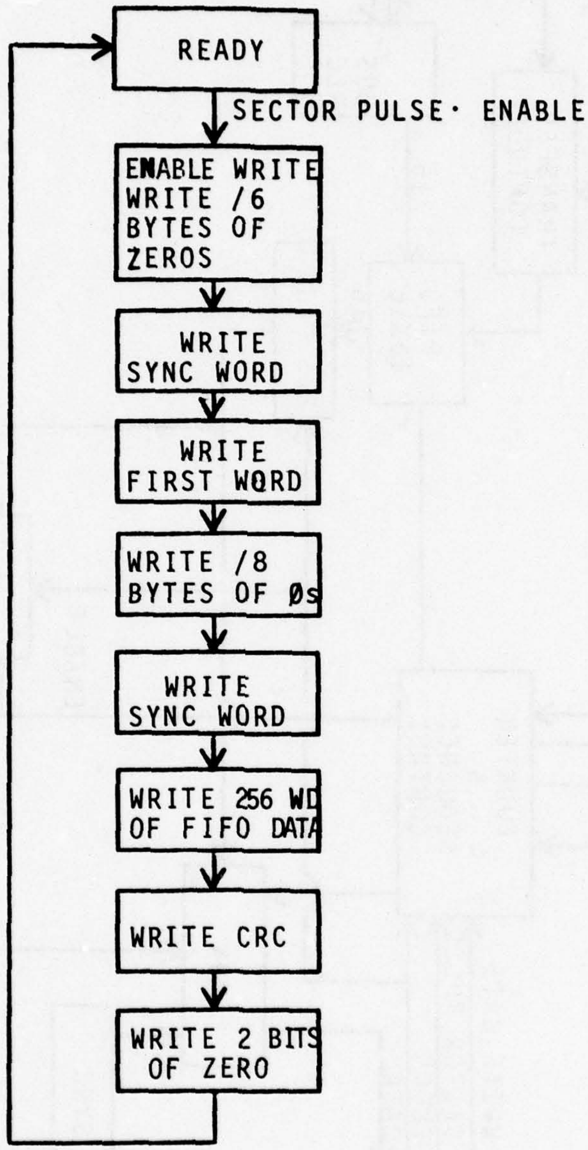


Figure 55 WRITE SEQUENCE

SECTION 8.0

FINDINGS, CONCLUSIONS AND RECOMMENDATIONS

8.0 FINDINGS, CONCLUSIONS AND RECOMMENDATIONS

The study addressed the feasibility of developing a security monitoring subsystem for the AN/GYQ-21(V), interactive analyst system, IAS. We find that it is feasible to protect data in the IAS, and that at least two and possibly three approaches will accomplish that objective. The approaches and their relative merits are described as follows:

- Data Base Guard - The data base guard approach allocates one central processing unit for each security compartment. Upon completion of the sign-in procedure, the user is switched to the computer that corresponds to the user's access privilege. A Secret privilege user will be switched to a Secret computer, and will be denied access to Top Secret records in this computer. Potential vulnerabilities are the data base guard (multiported disk controller) and the initial switching mechanism. Design of the data base guard was presented in Section 7. It was shown that the microprocessor that controls communications into and out of the data base has a small program that can be certified. This will prevent the insertion of trapdoors that would defeat the protection mechanism.
  The initial switching mechanism transfers the Secret user to the Secret IAS. It also represents a potential vulnerability. If the mechanism can be tricked into switching the Secret user to the Top Secret machine, the protection mechanism can be defeated. Fortunately, the microprocessor that performs the initial switching function has a small program that can be certified. We conclude that the potential vulnerabilities can be made impenetrable and as a result, the data base guard approach can be used as a basis to develop a secure IAS.
  Unfortunately, the approach is not without cost penalties. Multiple computers must be used (one for each compartment). Further, the switching mechanism must be added to the system. The data base guard approach is comprised of a microprocessor costing approximately $1500.
- Enciphered Record - The Enciphered Record approach presumes the processor is a hostile environment and applies the principles of RED-BLACK isolation the same as used in telecommunications. The records of the data base are enciphered, and even if accessed are useless without the ability to decipher them. The approach has appeal in that it is an elegant engineering solution to the problem of protecting data.
  Potential vulnerabilities of the system consist of the access control mechanism, RED processor, cracking the code, and stealing the key. The access control mechanism, like that of the data base guard, is comprised of a small microprocessor with certifiable software. Alternatively, the whole mechanism may be hard wired, thereby, eliminating the software problem. The problem of isolating the RED processor was addressed in Section 7, with the

conclusion that the BLACK processor can be totally isolated from the RED processor. We have assumed throughout that a block enciphering algorithm will be used that is impenetrable. For the initial design purposes, the NBS algorithm will be used, although it is recognized that this is unacceptable for military applications. We assume the Government will supply a military-acceptable block encryption algorithm. The final way the system is defeated is by stealing the key. This can be made impossible by a number of means. One way is to manually insert the key at the guard and design the guard in such a way that if a penetration is attempted, the guard will destroy the key before anyone can gain access to it.

The sections on RED-BLACK multiprocessing and the enciphered data base system were intended to demonstrate the complexity of making this approach invulnerable. On the basis of information set forth in the preceding sections, we conclude that the potential vulnerabilities of the enciphered record approach can be made impenetrable and, therefore, the enciphered record can be used as a basis for developing an impenetrable IAS.

The enciphered record approach, like the data base guard approach, does not come free of charge. An external access control mechanism must be developed comprised of microprocessors costing approximately $1500 per terminal. Further, because the records are enciphered and require processing, a RED processor must be added to the system. This will incur a modest addition in expense in most intelligence applications and can be accomplished by a mini-computer in most cases (one additional PDP-11, as opposed to several additional PDP-11's in the data base guard approach).

- Tag Approach - The tag approach to record security utilizes an enciphered tag to inform the exit guard of record classification. The approach is the simplest of the three and only requires an access control mechanism. Since the records are in the clear, RED-BLACK multiprocessing is not required.

  The approach is an effective measure against the threat of accidental disclosure. For this threat, it is the cheapest and will do the job. We, therefore, conclude that this approach is optimum against a benign threat.

  On the ability to withstand the attack by computer criminals, we are much less confident. As in the case of the enciphered record, potential vulnerabilities are access control mechanism, cracking the code, and stealing the key. But unlike the enciphered record approach with the data and records in the clear, the tag approach utilizes records that are in the clear. Therefore, they can be copied and dumped on a terminal or disguised as an error message and sent to the user. There are countermeasures to this vulnerability such as placing an exit guard at every terminal device and designing nonrecord communication containing an unforgeable tag, that describes the transaction as such. In practice, these may or may not be easy to implement.

There is some doubt and, therefore, we do not recommend a tag approach against the threat of malicious attack at this time.

Since, of the three possible approaches, one is at least questionable in its ability to withstand criminal attack, there remains only two possible alternatives for recommendation in its use in development of an engineering model. The data base guard approach appears the least vulnerable, but it is the most expensive. We, therefore, do not recommend it for implementation. There is a second reason for this. There is very little question of feasibility and building an engineering model of the data base guard approach would prove very little. Accordingly, we recommend that an engineering model of the enciphered record approach be developed as given in the next section.

Some experts will undoubtedly argue that building an admittedly special purpose data base system is not as productive an area of research as approaches to general purpose solutions, such as ongoing efforts to develop secure operating systems. Harris disagrees. First of all, a secure data base system would satisfy many military needs, not only in the intelligence community but also in command and control. Secondly, building special purpose machinery is an inductive approach to the general purpose problem. We have already observed that RED-BLACK multiprocessing is necessary to solve the secure data base problem. Quite possibly, a more general multiprocessing architecture will provide a basis for a multilevel secure computing system.

SECTION 9.0

SPECIFICATION OF A SECURE DATA BASE SYSTEM

9.0 SPECIFICATION OF A SECURE DATA BASE SYSTEM

9.1 Scope

9.1.1 Identification. This specification is for an Engineering Model of a Secure Data Base System (EMSDBS).

9.1.2 Functional Summary. The EMSDBS shall perform the following tasks:

- Sign-On and Sign-Off Procedures - This is an interactive communication between an operator and the system and provides the basis for preparation and termination of data base operational dialogue.
- Data Base Operational Dialogue - Provides the operator with the ability to retrieve, modify, add, and delete data base system records.
- RED-BLACK Multiprocessing - Provides secure computer-to-computer communication between two system computers.

The specification covers a model having certain security features that may be tested to establish that the enciphered record data base approach is in fact, secure.

9.2 Applicable Documents. The equipment and system software provided by the contractor shall be purchased from the Digital Equipment Corporation and is described in the following documents.

PDP-11/45 Processor Handbook
PDP-11 Peripherals Handbook
LSI-11 Processor Handbook
RSX-11M Volumes I through V
Introduction to RMS-11

9.3 Requirements. The EMSDBS shall be comprised of computing, peripheral equipment, system software and application programs. Its purpose is to provide data base communication to two or more operators in such a manner that the data base is secure. Security will be achieved by encrypting the records and data base and controlling the deciphering process.

The computing and peripheral equipment shall be as follows:

Equipment	QTY
BLACK processor, DEC, PDP-11/45	One
RED processor, LSI-11	One
Terminals for presentation and display of data base records	Two
Disk file - RK05	One
Exit guards - LSI-11	Two

Access control center - LSZ-11	One
Mailbox - RK05 initially	One

The system software shall be the RSX-11M operating system and RMS-11 records management system.

The major functions of the system are:

- Provide access to operators on and off the system
- Provide data base operational dialogue in the form of five commands; print, display, add, delete and change
- Secure communication between the RED and BLACK processors

9.3.1 Program Definition. The EMSDBS shall build upon the experience gained in the development of the Experimental, Enciphered Data Base System, described in Section 5 of this report. It will reuse concepts, design criteria, and software to the maximum extent possible. The EMSDBS will also incorporate the features of RED-BLACK multiprocessing described in Section 6.

Figure 56 shows a simplified block diagram of the EMSDBS. As in the Feasibility Model, only two terminals are used, each connected through a guard which remains transparent to communications once the user has signed on. The isolated, access control center commands the guards, verifies the identity of the user and detects any attempted sabotage. The BLACK processor contains the major portion of the software as well as the system software, RSX-11D and the data base management software RMS-11. It is a major processing element and controls all access to the data base. The latter is a disk file containing enciphered variable length records. The BLACK processor communicates to the RED processor by a mailbox. This design is described in Section 6, RED-BLACK Multiprocessing, of this report. The processing is required to demonstrate that it is feasible to achieve security while still deciphering records for processing.

Table 24 shows the utilization of equipment during the seven types of transactions that the user may execute from a given terminal. The transactions are described as follows:

- Sign-On - To gain access to the computing system, the user must sign-on. The user enters the code that identifies a sign-on procedure to the guard. The guard communicates control to the access control center in charge of executing the sign-on and sign-off process. The user is identified anc requests access to the data base. The access control center verifies identity by commanding the user to enter the password. After this is successfully accomplished, the access control center is satisfied with the user identity and checks the access privileges of that user, either Secret or Top Secret. If the user has Secret privilege only, that information is transferred from the access control center to the guard by way of a command stating that only Secret records will be passed to that terminal user. As shown in

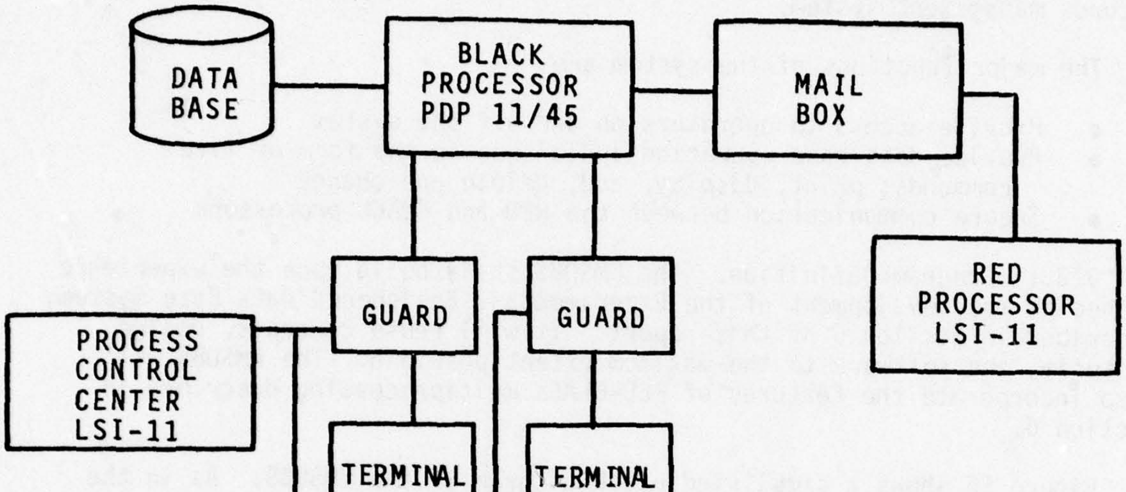


Figure 56 DATA CONTROL

Table 24. Function/Equipment Matrix

Function	Terminal	Guard	Access Control	Black Processor	Red Processor	Data Base
Sign-On	X	X	X			
Sign-Off	X	X	X			
Display	X	X		X		X
Print	X	X		X		X
ADD	X	X		X	X	X
Change	X	X		X		X
Delete	X	X		X		X

the table, the sign-on procedure involves the terminal and guard as to all seven transactions; however, the only additional equipment involved is the access control center.

- Sign-Off - This transaction simply terminates the user's time on the terminal and prevents a second user from gaining access with the same privileges as the preceding user. It also involves the same equipment as the sign-on procedure.
- Display - The command formats are described in Section 5 of this report. Briefly, display is a command that requests a particular record be presented to the terminal user. The request is via the terminal and guard, to the BLACK processor, which queries the data base and returns the record matching the query parameters. The record is returned to the guard for verification of the record classification. If the record is Secret, it is deciphered and passed on to the user. If the record is Top Secret, the user denied access the record is destroyed, and the access control center is notified that a potential breach in security has occurred.
- Print - This command is very similar to Display.
- ADD - This command adds a record to the data base. The format is described in Section 6 of this report. To test the feasibility of RED-BLACK processing, the add command is assumed, not only to result in a new record being added to the data base, but also that all terminal users be notified that the new record and an abstract of its contents have been added. The additional record created by the terminal user is enciphered by the guard before it is communicated to the BLACK processor. Therefore, the BLACK processor cannot process the record to obtain the abstract. The

BLACK processor takes two actions; it inserts the record into the data base and secondly, it communicates the enciphered record to the RED processor with instructions to calculate an unclassified abstract to the record for communication to the user. The communication between the BLACK processor and the RED processor is via the mailbox. The RED processor completes the abstract and returns the results to the BLACK processor for communication to the terminals. Table 24 shows that the ADD transaction utilizes all the equipment except the access control center.

- Change - This command results in the contents of a record being changed by the terminal user. The change record is enciphered by the guard before it is passed to the BLACK processor for replacement. The flow of information is from the terminal and guard, to the BLACK processor, to the data base.
- Delete - This transaction deletes the record from the data base. The information flow, as shown by the equipment utilized in Table 24 , is the terminal, the guard, the BLACK processor, and the data base.

Since the model is for engineering purposes only, the timing is of no consequence. Accordingly, no timing specifications are set forth.

The major functions of the computer program fall into four major categories as follows:

- Guard Program - The Guard Program is described in Paragraph 5.2 of this document.
- Access Control Center Program (ACC) - Figure 57 shows the overall program for the access control center. After it is initialized, the program checks the incoming lines from the two guards to detect the sign-on procedure. If a sign-on procedure has been completed by an operator, the ACC program proceeds to verify the identity of the user, look up the access privileges, and command the guard to pass records within that access privilege. If there are no sign-on requests, the program checks for sign-off procedures. If such a procedure is detected, the access control center clears the guard of its current access privileges, reinitializes, and returns to the beginning of the program. If no sign-off procedure is detected, the program proceeds to check for errors. An error is encountered when a Secret terminal has requested a Top Secret record. If no errors are detected, the program returns to its beginning point. If an error is detected, a processing algorithm is entered before returning control to the beginning of the program.
- BLACK Processor Program - The BLACK Processor program has been described in Paragraph 5.3 of this document.
- RED Processor - Figure 58 shows the flow diagram of the RED Processor program after initialization. The program continuously searches the mailbox for an ADD transaction from a user. This is the only transaction that requires the aid of the RED

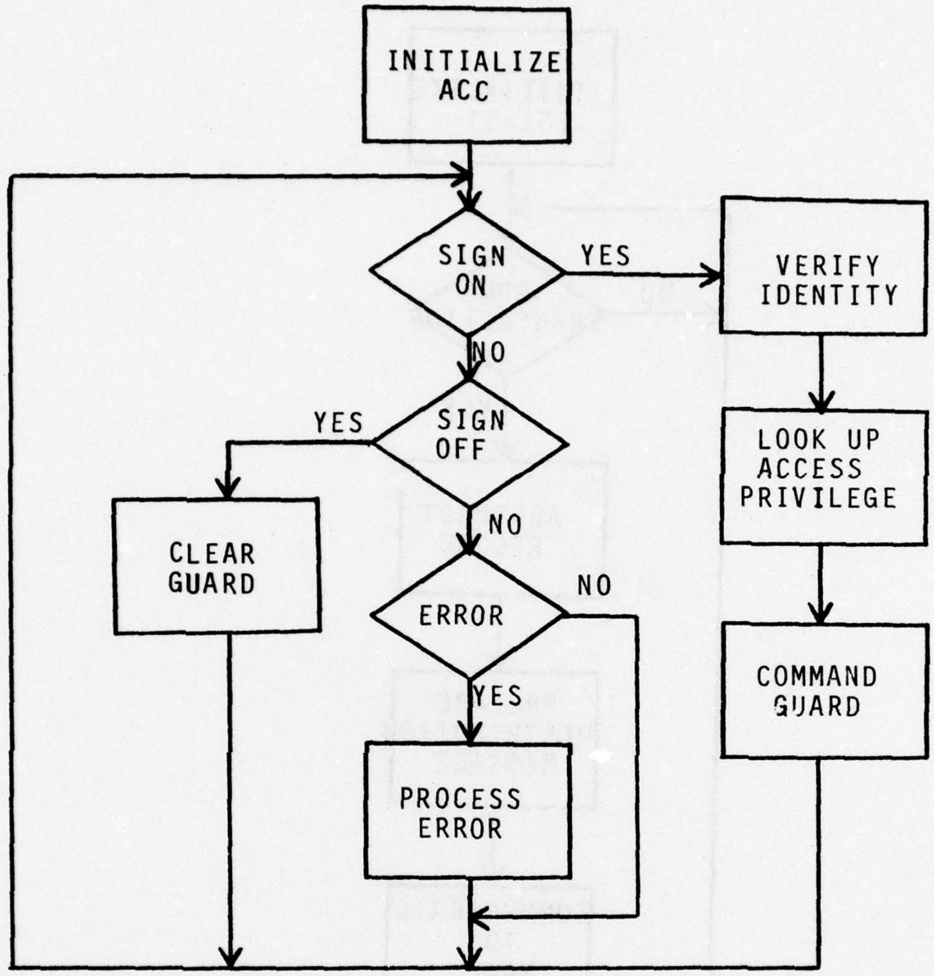


Figure 57 ACCESS CONTROL CENTER

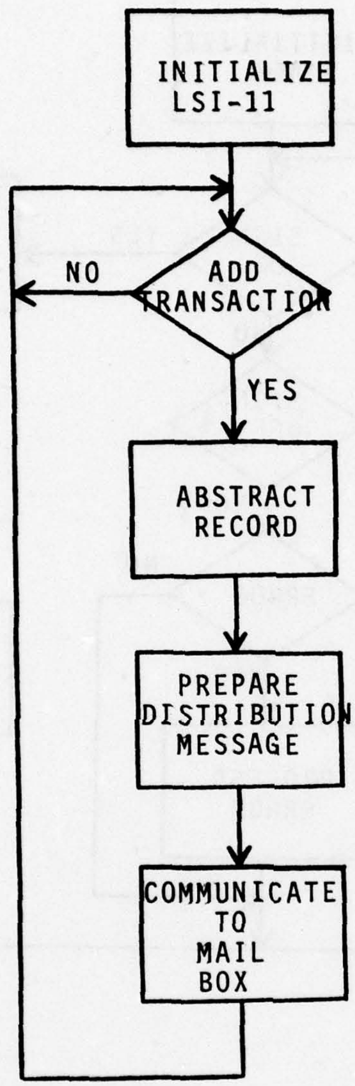


Figure 58 RED PROCESSOR PROGRAM

processor. When an ADD transaction is encountered, the record is abstracted and messages are prepared for distribution to the two terminals. The messages are communicated to the mailbox for distribution and the program returns to its beginning point.

9.3.2 Detailed Functional Requirements. The inputs, outputs, and processing for the four types of programs are:

- Guard Program - Reference Paragraph 5.2.1 of this report.
- Access Control Center Program - The inputs, outputs, and processing are to be determined.
- BLACK Processor - Reference Paragraph 5.3.1 of this report. These programs will also interface to RSX-11M. See "Introduction to RSX-11M."
- RED Processor - Inputs, outputs, and processing to be determined.

9.3.3 Adaptation. The details on the data requirements f the secure data base system are described in "Introduction to RSX-11M."

9.4 Quality Assurance Provisions

9.4.1 Introduction. The size of the programs dictates that development will take place on a set of modules integrated into subprograms and programs. Examples of the development work and listings are shown in the appendix. The guard programs, access control center programs, and red processor programs will be developed as separate entities. The black processor programs are more complex, requiring interface with RSX-11M and RMS-11. System tests will integrate all of the programs into an operational system.

9.4.2 Test Requirements. Testing will proceed at the module, subprogram and program level. Final tests will be based upon interactive dialogue between the data base and the operator.

9.4.3 Acceptance Test Requirements. The operations system will be demonstrated by a user/query response, as specified in the interactive dialogue described in Section 5 of this report. The contractor will prepare and submit an acceptance test for the buyer's approval. The buyer will witness the acceptance.

Security testing will begin following acceptance of the FMSDBS as an operational system. A test plan will be prepared to examine the security of the data bus system.

9.5 Preparation For Delivery. The EMSDBS will be installed at the contractor's facility where acceptance testing will occur. Demonstrations of feasibility will also occur at the contractor's site. It is not anticipated that the EMDBS would be shipped to a Government facility.

APPENDIX A

PROGRAM LISTINGS

COPY AVAILABLE TO DDC DOES NOT
PERMIT FULLY LEGIBLE PRODUCTION

```
$JOB DATABASE BLD ON DK1
DATE:-17-AUG-77
TIME:-00:23:08
$RUN PIP
PIP V10-03A
*COMPOL.001/DE

*RPN.SRC/DE

*COMPOL.001<COMPOL.ISS

*RPN.SRC<BI:/FA

$EOD

*
$RUN JOVIAL
```

```
JOVIAL ISS                            *   HARRIS JOVIAL COMPILER
                                          ---VERSION -3A           PAGE    1
 '' THIS PROGRAM READS EVENT RECORDS FROM THE CARD READER AND          ''          1
 ''    STORES THEM ON THE DATABASE DISK FOR USE BY THE ISS SYSTEM     ''          1
       I = 0 $                                                                     1
       START'TRACE $                                                               2
          DIRECT    $                                                              2
          .EVEN                                                                    2
          .MACRO  STMT NO, ?A1                                                     2
          .GLOBL  STMTNO                                                           2
          MOV     #NO,STMTNO             ;STORE STATEMENT NUMBER                   2
          .GLOBL  TRACE                                                            2
          JSR     R5,TRACE                                                         2
          BR      A1                                                               2
          .WORD   STMTNO                                                           2
A1:                                                                                2
          .ENDM                                                                    2
          JOVIAL                                                                   2
       DIRECT $                                                                    2
OVER:     JSR     R5,RCD              ; READ NAME OF EVENT                         2
       JOVIAL                                                                      2
       IF ICARD($0$) EQ ATCHR  $ STOP $                                            2
       CSMOV (30,@NAME,@CARD,1) $                                                  4
       READ (8,RCDF1) LATITUDE,LONGITUDE,EVENTTYPE,COUNTRY,TIME,REPORT'NO          5
                 $                                                                 5
       DIRECT $                                                                    6
          JSR     R5,RCD              ; READ CASE'NO,COMMENTS,CTAG'E               6
       JOVIAL                                                                      6
       CSMOV (8,@CTAG'E,30,@COMMENTS,12,@CASE'NO,@CARD,3) $                        6
 '' NOW EVENT IS STORED IN COMPOOL, TRANSFER TO DISK ''                            7
       DKTRN (@EVENT) $                                                            7
       GOTO OVER $                                                                 8
RCDF1. FORMAT (2F10.4,2I1,F10.4,I10) $                                             9
       DIRECT $                                                                   10
          .MCALL  .INIT,.TRAN,.WAIT                                               10
          .GLOBL  DKTRN                                                           10
          MSTK = %5                                                               10
DKTRN:                                                                            10
          .INIT   #DK1                ; INIT LINK BLOCK                           10
          LDF     (MSTK)+,AC0         ; CONVERT ADDR OF EVENT TO FIXED            10
          STCFI   AC0,-(MSTK)         ;                                           10
          TST     (MSTK)+                                                         10
          MOV     (MSTK)+,TRNBLK+2    ; STORE ADDR OF EVENT IN TRNBLK             10
          .TRAN     #DK1,#TRNBLK      ; WRITE TO DISK                             10
          .WAIT     #DK1              ; WAIT UNTIL I/O IS DONE                    10
          INC     TRNBLK              ; INC BLOCK # FOR NEXT RECORD               10
          RTS     PC                                                              10
          .WORD   0                   ; ERROR RETURN                              10
DK1:      .WORD   0                   ; LINK POINTER                              10
          .RAD50  /   /               ; LOGICAL NAME OF DATASET                   10
          .BYTE   1,1                 ; NUMBER OF WORDS, UNIT #                   10
          .RAD50  /DK/                ; PHYSICAL DEVICE NAME                      10
TRNBLK:   .WORD   2400.               ; STARTING BLOCK #                          10
          .WORD   0                   ; STARTING ADDR IN CORE                     10
          .WORD   256.                ; NUMBER OF WORDS                           10
          .WORD   2                   ; WRITE                                     10
          .WORD   0                   ; FOR MONITOR USE                           10
```

```
         .GLOBL  CSMOV                                                  10
CSMOV:           ; ROUTINE TO MOVE A CHAR STRING FROM ICARD             10
                 ; TO THE DESIRED COMPOOL ITEM                          10
         MOV     R0,-(SP)                                               10
         MOV     R1,-(SP)                                               10
         MOV     R2,-(SP)                                               10
         MOV     R3,-(SP)                                               10
         JSR     PC,GNSE                                                10
         MOV     (MSTK)+,R0         ; # OF STRINGS TO MOVE              10
         JSR     PC,GNSE            ; GET NEXT STACK ENTRY              10
         MOV     (MSTK)+,R1         ; LOCATION OF STRINGS               10
1$:                                                                     10
         JSR     PC,GNSE            ; GET NEXT STACK ENTRY              10
         MOV     (MSTK)+,R2         ; ADDR TO MOVE TO                   10
         JSR     PC,GNSE            ; GET NEXT STACK ENTRY              10
         MOV     (MSTK)+,R3         ; LENGTH OF STRING                  10
2$:      MOVB    (R1)+,(R2)+        ; MOVE DATA CHAR                    10
         SOB     R3,2$                                                  10
         SOB     R0,1$                                                  10
         MOV     (SP)+,R3                                               10
         MOV     (SP)+,R2                                               10
         MOV     (SP)+,R1                                               10
         MOV     (SP)+,R0                                               10
         RTS     PC                                                     10
         .GLOBL  GNSE                                                   10
GNSE:    LDF     (MSTK)+,AC0                                            10
         STCFI   AC0,-(MSTK)                                            10
         TST     (MSTK)+                                                10
         RTS     PC                                                     10
         JOVIAL                                                         10
         END PROGRAM $
```

DATA BASE REFERENCE LISTING

S	5	13	0	0
CARD	0	2	1	40
ICARD	0	14	0	8
ATCHR	75	9	0	8
EVENT	6501	2	1	80
NAME	6501	14	0	240
LATITUDE	6520	11	0	32
LONGITUDE	6522	11	0	32
EVENTTYPE	6524	9	0	8
COUNTRY	6524	9	8	8
TIME	6525	9	0	32
REPORT NO	6527	9	0	16
CASE NO	6530	9	0	96
COMMENTS	6536	9	0	240
CTAG E	6555	9	0	64
I	1	13	0	16

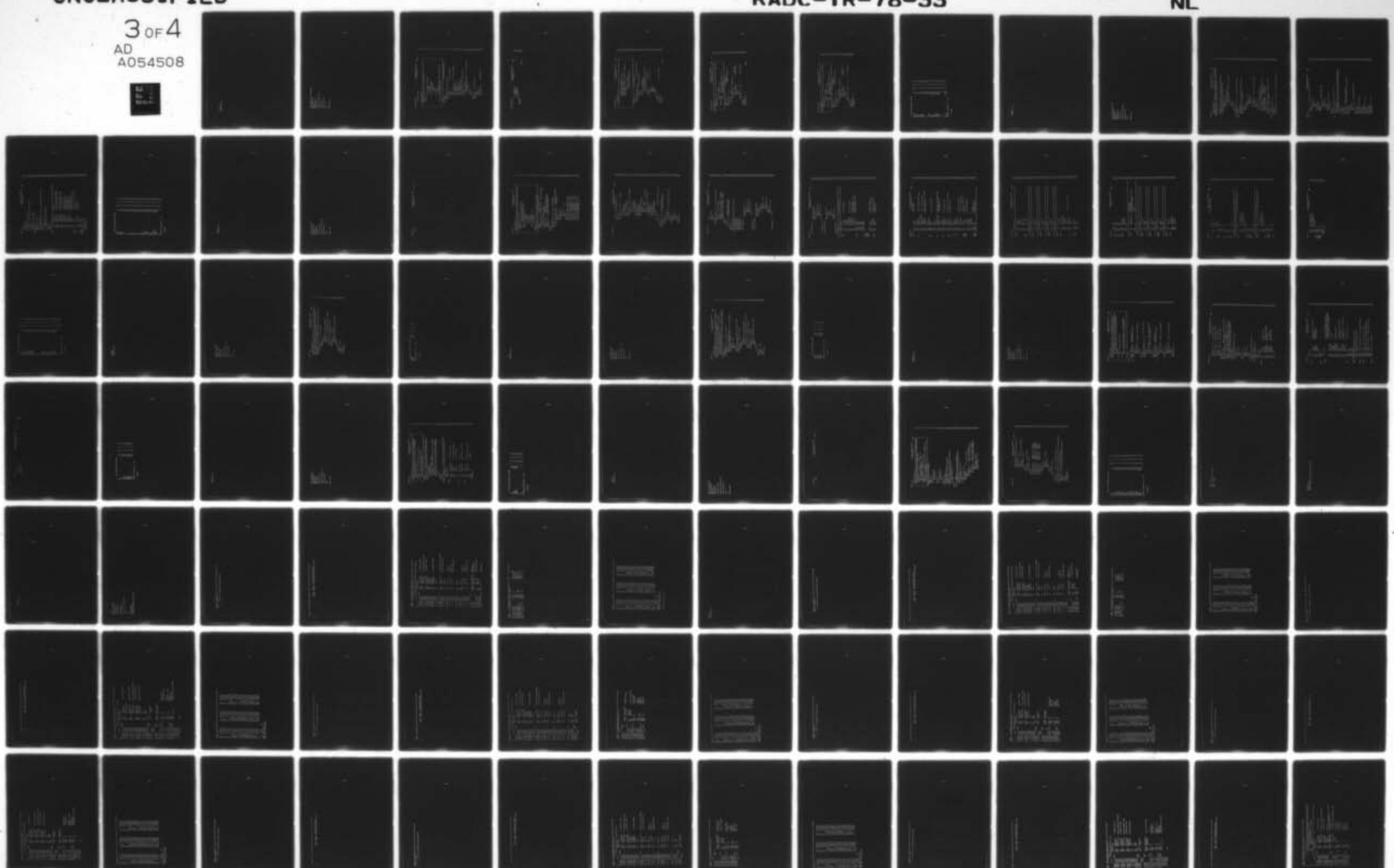
COMPOOL SPACE USED = 1098

0 ERRORS

AD-A054 508 HARRIS CORP MELBOURNE FLA ELECTRONIC SYSTEMS DIV F/G 9/2
INTELLIGENCE SECURITY SUBSYSTEM.(U)
MAR 78 F ANDERS, W MALL, R MCGILL F30602-76-C-0445

UNCLASSIFIED RADC-TR-78-33 NL

3 OF 4
AD
A054508



```
$FINISH
TIME:-00:25:32
```

```
$JOB COMPILE ISS SUBROUTINES
DATE:-17-AUG-77
TIME:-00:26:26
$RUN PIP
PIP V10-03A
*COMPOL.001/DE

*COMPOL.001<COMPOL.ISS

*RPN.SRC/DE

*RPN.SRC<BI:/FA

$EOD

*
$RUN JOVIAL
```

```
ISS SUBROUTINES                          *   HARRIS JOVIAL COMPILER
                                             ---VERSION -3A             PAGE    1
     'PROGRAM ISSUBS          $                                                        1
                                                                                       2
                                                                                       2
     LCP SUBROUTINE  $                                                                 2
''-------------------------------------------------------------------------''          3
''    ROUTINE TO STORE OR FETCH A COMPOOL ENTRY                             ''          3
''                                                                          ''          3
''    INPUTS ARE :                                                          ''          3
''          FUN = FUNCTION TO PERFORM                                       ''          3
''              = 0 FETCH                                                   ''          3
''              = 1 STORE                                                   ''          3
''          IERR = ERROR CONDITION                                          ''          3
''               = 0 NO ERROR                                               ''          3
''               = 1 ERROR                                                  ''          3
''          A    = TYPE                                                     ''          3
''          B    = FR                                                       ''          3
''          C    = NB                                                       ''          3
''          D    = ADDRESS                                                  ''          3
''          TAG = ADDRESS OF BYTE STORAGE, NTAG = LENGTH                    ''          3
''    OUTPUTS ARE THE ABOVE ITEMS SET                                       ''          3
''                                                                          ''          3
''-------------------------------------------------------------------------''          3
     IERR = 0        $              '' RESET ERROR CONDITION ''                        3
     IF FUN EQ 1 $                                                                     4
        THEN        $                                                                  5
        CALL SCP $  '' SEARCH COMPOOL   ''                                             6
        IF IERR EQ 0 $                                                                 7
           THEN                     '' FOUND SET ERROR ''                              8
           IERR = 1 $                                                                  8
           ELSE $                                                                      9
           IERR  =  0 $             '' PUT DATA IN COMPOOL      ''                    10
           TYP  (SINENTS)= A $                                                        11
           REF (SINENTS)= 0 $                                                         12
           IFR (SINENTS)= B $                                                         13
           INB (SINENTS)= C $                                                         14
           ADR (SINENTS)= D $                                                         15
           POINT (SINENTS)= NSYM $                                                    16
           NCHAR (SINENTS)= NTAG $                                                    17
           INENT = INENT + 1 $ ''  STEP INDEX  ''                                     18
           I = 0 $                                                                    19
           DO WHILE I LS NTAG $                                                       20
              SYM(SNSYMS) = TAG(SIS) $  '' MOVE IN SYMBOL ''                          21
              NSYM = NSYM +1 $                                                        22
              I = I + 1 $                                                             23
           END DO $                                                                   24
           END IF    $                                                                25
           RETURN    $                                                                26
        ELSE $                                                                        27
        CALL SCP $  '' SEARCH COMPOOL   ''                                            28
        IF IERR EQ 1    $                                                             29
           THEN                                                                       30
           RETURN $                                                                   30
           ELSE    $                                                                  31
           A = TYPE(SIS) $      '' RETRIEVE DATA  ''                                  32
           REF(SIS) = 1 $                                                             33
```

ISS SUBROUTINES   *   HARRIS JOVIAL COMPILER
                                    ---VERSION -3A   PAGE   2

```
                B = IFR(SIS) $
                C = INR(SIS) $
                D = ADR(SIS) $
                END IF $
          END IF $
          RETURN $
                                                       PAGE $
```

```
ISS SUBROUTINES                          *   HARRIS JOVIAL COMPILER
                                             ---VERSION -3A          PAGE     3
        SCP SUBROUTINE  $                                                          41
''--------------------------------------------------------------------''           42
''                                                                    ''           42
''              SEARCH COMPOOL SUBROUTINE                             ''           42
''              FUNCTION IS TO SEARCH THE COMPOOL                     ''           42
''              FOR A MATCHING ITEM                                   ''           42
''              INPUTS ARE THE SAME AS THE COMPOOL SUBROUTINE         ''           42
''              OUTPUTS ARE                                           ''           42
''                        IERR = 0 = FOUND                            ''           42
''                        IERR = 1 = NOT FOUND                        ''           42
''                        I = POINTER TO FOUND ENTRY                  ''           42
''                                                                    ''           42
''                                                                    ''           42
''--------------------------------------------------------------------''           42
        IERR = 1 $            '' SET ERROR RETRUN  ''                              42
        I = 0 $                                                                    43
        DO UNTIL I GQ IMENT  $            '' LOOP THROUGH COMPOOL  ''              44
             IF NCHAR($I$) EQ NTAGS                                                45
                  THEN $      '' SEE IF TAG MATCHES  ''                            46
                  J = 0 $                                                          47
                  L = 0 $                                                          48
                  DO WHILE J LS NCHAR($I$)$                                        49
                       K = J + POINT($I$) $                                        50
                       IF SYM($K$) NQ TAG($J$)$ THEN$ L = 1$                       51
                       ELSE $ END IF $                                             54
                  J = J + 1 $           '' STEP INDEX  ''                          56
                  END DO $                                                         57
                  IF L EQ 0$            '' 0 MEANS FOUND  ''                       58
                       THEN$                                                       59
                       IERR = 0 $                                                  60
                       RETURN $                                                    61
                       ELSE$                                                       62
                  END IF $                                                         63
            ELSE $                                                                 64
             END IF $                                                              65
        I = I + 1  $                    ''  STEP LOOP INDEX   ''                   66
        END DO $                                                                   67
        RETURN $                                                                   69
                                                              PAGE$                70
```

```
  ''----------------------------------------------------------------------''    71
OCII SUBROUTINE  $                                                               71
 ''      ROUTINE TO CONVERT AN OCTAL  STRING TO AN OCTAL  BINARY NUMB ''        72
 ''      INPUTS ARE:                                                  ''        72
 ''                ICARD  = IMAGE                                     ''        72
 ''                ICOL   = STARTING COLUMN                           ''        72
 ''      OUTPUTS ARE:                                                 ''        72
 ''                ICOL   = POINTS TO FIRST NONOCTAL    COL           ''        72
 ''                SUM    = CONVERTED RESULT                          ''        72
  ''----------------------------------------------------------------------''    72
         ITEM  OCIE   I 1 U $     '' END INDICATOR  ''                          73
         SUM = 0 $                         '' CLEAR SUM ''                      74
         OCIE   =  0 $                     '' RESET END INDICATOR ''            75
         DO WHILE OCIE   EQ 0 $                                                 76
             I = 0          $                                                   77
             DO WHILE  I LS  8 $                                                78
                 IF ICARD($ICOL$) EQ IDEC($I$) $                                79
                     THEN                                                       79
                     SUM = SUM*8  + I          $  '' NEW SUM ''                 80
                     ICOL = ICOL+1 $              '' STEP INDEX ''              81
                     ELSE $                                                     82
                     I = I + 1  $                                               83
                 END IF $                                                       84
                 IF  I GQ  8 $  OCIE   = 1 $      ''SET DONE FLAG ''            86
             END DO $                                                           87
         END DO $                                                               88
         RETURN $                                                               89
         END SUBROUTINE  $                                    PAGES             90
```

```
ISS SUBROUTINES                          *    HARRIS JOVIAL COMPILER
                                              ---VERSION -3A             PAGE     5
  ''----------------------------------------------------------------------''             91
UDII SUBROUTINE  $                                                                       91
  ''      ROUTINE TO CONVERT A DECIMAL STRING TO A DECIMAL BINARY NUMB ''                92
  ''      INPUTS ARE:                                                  ''                92
  ''               ICARD  = IMAGE                                      ''                92
  ''               ICOL   = STARTING COLUMN                            ''                92
  ''      OUTPUTS ARE:                                                 ''                92
  ''               ICOL   = POINTS TO FIRST NON-DECIMAL COL            ''                92
  ''               SUM    = CONVERTED RESULT                           ''                92
  ''----------------------------------------------------------------------''             92
        ITEM  UDIE   I 1 U $     '' END INDICATOR  ''                                    92
        SUM = 0 $                       '' CLEAR SUM ''                                  93
        UDIE   =  0 $                   '' RESET END INDICATOR ''                        94
        DO WHILE UDIE   EQ 0 $                                                           95
             I = 0          $                                                            96
             DO WHILE  I LS 10 $                                                         97
                  IF ICARD($ICOL$) EQ IDEC($I$) $                                        98
                       THEN                                                              99
                       SUM = SUM*10 + I          $  '' NEW SUM ''                        99
                       ICOL = ICOL+1 $              '' STEP INDEX   ''                  100
                       ELSE $                                                           101
                       I = I + 1  $                                                     102
                  END IF $                                                              103
                  IF  I GQ 10 $  UDIE   = 1 $      ''SET DONE FLAG ''                   104
             END DO $                                                                   106
        END DO $                                                                        107
        RETURN $                                                                        108
        END SUBROUTINE  $                                                               109
        END PROGRAM       $                                                             110
```

DATA BASE REFERENCE LISTING

S	5	13	0	0
ICARD	0	14	0	8
IDEC	67	14	0	8
ICOL	106	9	0	16
TYPE	112	9	0	16
SUM	115	9	0	16
FUN	117	9	0	16
IERR	120	9	0	16
NTAG	122	9	0	16
NSYM	123	9	0	16
A	124	9	0	16
B	125	9	0	16
C	126	9	0	16
D	127	9	0	16
INENT	131	9	0	16
TAG	170	14	0	8
TYP	2057	9	0	4
REF	2057	9	4	1
IFB	2057	9	5	5
INB	2057	9	10	6
NCHAR	3043	9	0	5
POINT	3043	9	5	11
ADR	4027	9	0	16
SYM	5013	14	0	8
I	1	13	0	16
J	2	13	0	16
K	3	13	0	16
L	4	13	0	16
OCTE	6746	9	0	1
UDIE	6746	9	0	1

COMPOOL SPACE USED = 1123

0 ERRORS

SFINISH
TIME:-00:31:37

```
$JOB COMPILE COMPOL READ
DATE:-17-AUG-77
TIME:-00:31:49
$RUN PIP
PIP V10-03A
*COMPOL.001/DE

*RPN.SRC/DE

*COMPOL.001<COMPOL.ISS

*RPN.SRC<BI:/FA

$EOD

*
$RUN JOVIAL
```

```
      LCMPL SUBROUTINE $
   '' SUBROUTINE LCMPL - LOAD COMPOOL INTO SHARED MEMORY.  THIS PROGRAM ''
   '' READS THE DISK AND FINDS THE FILE - COMPOL.001 - AND LOADS IT INTO''
   '' THE PROPER PLACE IN  THE COMPOOL IN SHARED MEMORY                 ''

      I = 0 $
      DSKRW(@DISK'BUF,2,512) $ '' READ MFD BLOCK INTO BUFFER ''
      I = 1 $ '' SET INDEX TO 1, SKIP MFD LINK PTR ''
      DO UNTIL DISK($I$) EQ 0 $ '' CHECK ALL UIC S ''
          IF DISKB($I*2$) EQ 2 $
              THEN $
              IF DISKB($I*2+1$) EQ 1 $
                  THEN $  '' FOUND 1,2 UIC ''
                  J = DISK($I+1$) $ '' SAVE UFD START BLK ''
                  ELSE $
              END IF $
              ELSE $
              I = I + 4 $
          END IF $
      END DO $
NEXT. DSKRW(@DISK'BUF,J,512) $ '' READ UFD START BLOCK ''
      I = 1 $
NXT1. TI = 0 $
      TL = @DISK($I$) $
      CHKFN(@TL,@TI) $ '' CHECK FILENAME FOR COMPOL.001 ''
      IF TI GR 0 $ '' THEN NO MATCH ''
          THEN $
          I = I + 9 $
          IF I GR 256 $
              THEN $
              IF DISK($0$) EQ 0 $
                  THEN $
                  CALL ERRM $
                  ELSE $
              END IF $
              J = DISK($0$) $ '' SET J TO NEXT UFD BLOCK ''
              GOTO NEXT $
              ELSE $
              GOTO NXT1 $
          END IF $
          ELSE $ '' FOUND CORRECT ENTRY ''
          I = I + 4 $'' FOUND CORRECT ENTRY ''
          EBP = DISK($I$) $    '' BYTE COUNT OF LAST BLOCK ''
          SBLK = DISK($I+1$) $ '' START BLOCK ''
          LNGTH = DISK($I+2$) $  '' LENGTH ''
          EBLK = DISK($I+3$) $  '' END BLOCK ''
      END IF $
      '' BEGIN TO LOAD FILE COMPOL.001 INTO SHARED MEMORY ''
      TJ = 0 $  '' INDEX TO FMTBUF ''
      J = SBLK $
OVER.
      DSKRW(@DISK'BUF,J,512) $
      TK = 0 $
      L = 511 $
      IF DISK($0$) EQ 0 $ L = EBP-1 $ '' BYTE COUNT OF LAST BLOCK ''
```

```
OVR.  IF TK GQ L $ '' END OF SECTOR OR FILE ''                                   52
           THEN $                                                                53
           IF DISK($0$) EQ 0 $                                                   54
                THEN $                                                           55
                GOTO OUT $                                                       56
                ELSE $                                                           57
           END IF $                                                              58
           J = DISK($0$) $                                                       59
           GOTO OVER $                                                           60
           ELSE $                                                                61
           IF TJ EQ 37 $  '' PROCESS CARD IMAGE ''                               62
                THEN $                                                           63
                CALL PLIMG $                                                     64
                TJ = 0 $                                                         65
                ELSE $                                                           66
           END IF $                                                              67
           FMTBF($TJ$) = DISKB($TK$) $                                           68
           TJ = TJ + 1 $                                                         69
           TK = TK + 1 $                                                         70
      END IF $                                                                   71
      GOTO OVR $                                                                 72
OUT.       '' FINISH READING AND STORING FILE ''                                 73
      RETURN $                                                                   73
      PLIMG SUBROUTINE $                                                         74
      '' THIS SUBROUTINE TAKES THE CARD IMAGE THAT HAS BEEN LOADED     ''        75
      '' INTO FMTBUF AND LOADS THAT COMPOL DATA INTO THE SCARED MEMORY''         75
      T1 = TT1 $                                                                 75
      T2 = TT2 $                                                                 76
      T3 = TT3 $                                                                 77
      T4 = TT4 $                                                                 78
      T5 = TT5 $                                                                 79
      TM = 10 $  '' INDEX FOR 1ST T5 STRING ''                                   80
      CALL LOADC $  '' LOADS DEC STRING INTO CARD FOR CDCON ''                   81
      CALL CDCON $                                                               82
      IF DCR EQ 0 $                                                              83
           THEN $                                                                84
           CALL ERRM $                                                           85
           ELSE $                                                                86
           NTAG = SUM $ '' STORE CONVERTED RESULT ''                             87
      END IF $                                                                   88
      TM = 15 $                                                                  89
      CALL LOADC $                                                               90
      CALL CDCON $                                                               91
      IF DCR EQ 0 $ CALL ERRM $                                                  92
      A = SUM $                                                                  94
      TM = 20 $                                                                  95
      CALL LOADC $                                                               96
      CALL CDCON $                                                               97
      IF DCR EQ 0 $ CALL ERRM $                                                  98
      B = SUM $                                                                  100
      TM = 25 $                                                                  101
      CALL LOADC $                                                               102
      CALL CDCON $                                                               103
      IF DCR EQ 0 $ CALL ERRM $                                                  104
      C = SUM $                                                                  106
```

```
      TN = 30 $                                                        107
      CALL LOADC $                                                     108
      CALL CDCON $                                                     109
      IF DCP EQ 0 $ CALL ERRM $                                        110
      D = SUM $                                                        112
      '' NOW HAVE VALUES LOADED FOR LCP TO WORK ''                     113
      FUN = 1 $                                                        113
      CALL LCP $                                                       114
      IF NTAG EQ 1 $ GOTO LOOP2 $                                      115
      RETURN $                                                         117
LOOP2.IF T1 EQ AT $ GOTO OUT $                                         118
      RETURN $                                                         120
      ERRM SUBROUTINE $                                                121
      RETURN $                                                         122
      LOADC SUBROUTINE $                                               123
      '' THIS ROUTINE LOADS A DECIMAL STRING INTO CARD FOR CDCON ''    124
      ICOL = 0 $                                                       124
      TN = 0 $                                                         125
      DO UNTIL TN GQ 5 $                                               126
          ICARD(STNS) = FMTBF(STN+TMS) $                               127
          TN = TN + 1 $                                                128
      END DO $                                                         129
      RETURN $                                                         131
      DIRECT $                                                         132
        .GLOBL  CHKFN                                                  132
CHKFN:  ; THIS ROUTINE COMPARES 3 RAD50 WORDS WITH FILENAME COMPOL.001 132
        MOV     R0,-(SP)           ; SAVE REG                          132
        MOV     R1,-(SP)           ;                                   132
        LDF     (MSTK)+,AC0        ; STORE AND CONVERT ADDRESS         132
        STCFI   AC0,-(MSTK)        ; TO RETURN RESULT, 0=F,1=T         132
        TST     (MSTK)+                                                132
        MOV     (MSTK)+,R0         ; STORE ADDR IN R0                  132
        LDF     (MSTK)+,AC0        ; STORE AND CONVERT ADDRESS         132
        STCFI   AC0,-(MSTK)        ; OF 1ST RAD5/ WORD                 132
        TST     (MSTK)+                                                132
        MOV     (MSTK)+,R1         ; STORE ADDR IN R1                  132
        CMP     FLNM1,(R1)         ; COMPARE 1ST 3 CHAR                132
        BNE     NMTCH              ; BR IF NO MATCH                    132
        CMP     FLNM2,2(R1)        ; COMPARE 2ND 3 CHAR                132
        BNE     NMTCH              ; BR IF NO MATCH                    132
        CMP     FLNM3,4(R1)        ; COMPARE 3RD 3 CHAR                132
        BNE     NMTCH              ; BR IF NO MATCH                    132
        MOV     #1,@R0             ; SET RESULT TO 1 FOR TRUE          132
        BR      ND                                                     132
NMTCH:  CLR     @R0                ; SET TESULT TO 0 FOR FALSE         132
ND:                                                                    132
        MOV     (SP)+,R1           ; RESTORE REG                       132
        MOV     (SP)+,R0           ;                                   132
        RTS     PC                 ; RETURN                            132
FLNM1:  .RAD50  /COM/                                                  132
FLNM2:  .RAD50  /POL/                                                  132
FLNM3:  .RAD50  /001/                                                  132
       JOVIAL                                                          132
       END SUBROUTINE $                                                132
       END PROGRAM $                                                   133
```

```
DATA BASE REFERENCE LISTING
S                              5   13    0    0
ICARD                          0   14    0    8
AI                            54   14    8    8
ICOL                         106    9    0   16
SUM                          115    9    0   16
FUN                          117    9    0   16
NTAG                         122    9    0   16
A                            124    9    0   16
B                            125    9    0   16
C                            126    9    0   16
D                            127    9    0   16
T1                           170    9    0   16
T2                           171    9    0   16
T3                           172    9    0   16
T4                           173    9    0   16
T5                           174    9    0   16
DCR                          573    9    0   16
TI                           604    9    0   16
TJ                           605    9    0   16
TK                           607    9    0   16
TL                           610    9    0   16
TM                           611    9    0   16
TN                           612    9    0   16
SBLK                         631    9    0   16
ERP                          632    9    0   16
LNGTH                        633    9    0   16
EBLK                         634    9    0   16
FMTBF                        644   14    0    8
TT1                          645    9    0   16
TT2                          646    9    0   16
TT3                          647    9    0   16
TT4                          650    9    0   16
TT5                          651    9    0   16
DISK BUF                    5777    2    1  256
DISKB                       5777   14    0    8
DISK                        5777    9    0   16
I                              1   13    0   16
J                              2   13    0   16
L                              4   13    0   16
COMPOOL SPACE USED =  1143

  0 ERRORS
```

```
$FINISH
TIME:-00:36:53
```

```
$JOB CCXEC COMPILE
DATE:-22-AUG-77
TIME:-00:00:15
$RUN PIP
PIP V10-03A
#RPN.SRC/DE

#COMPOOL.001/DE

#COMPOOL.001<COMPOOL.ISS

#RPN.SRC<BI:/FA

$EOD

#
$RUN JOVIAL
```

```
      CCXEC SUBROUTINE $                                                          2
''-----------------------------------------------------------------------''      3
''    CODE EXECUTION - CARRIES OUT THE PRESCRIBED OPERATIONS OF THE      ''      3
''       GENERATED POLISH STRING                                         ''      3
''    INPUTS ARE:                                                        ''      3
''       POLSTK = CONTAINS POLISH TARGET STRING                          ''      3
''                (ACCESSED BY PNTRY(PINDX) )                            ''      3
''         MSTK = USED AS OPERAND STACK HERE                             ''      3
''        MVTYP = MAIN ACTION TYPE                                       ''      3
''              = 1 = DISPLAY                                            ''      3
''              = 2 = PRINT                                              ''      3
''              = 3 = DELETE                                             ''      3
''              = 4 = ADD                                                ''      3
''              = 5 = CHANGE                                             ''      3
''        MSTYP = MAIN SUBJECT TYPE                                      ''      3
''              = 1 = EVENT                                              ''      3
''              = 2 = REPORT                                             ''      3
''    OUTPUTS ARE:                                                       ''      3
''       RESULTS OF CODE EXECUTION (DEPENDENT ON COMMAND)                ''      3
''-----------------------------------------------------------------------''      3
      IF MSTYP EQ 1 $           '' SET DISK SECTOR BASE ''                        3
         THEN $                 '' EVENT ''                                       4
         BCRECD = NENT(EVENT) * NWSEN(EVENT) $ ''BYTE COUNT-EVENT''               5
         K = NOEVNT $                                                             6
      J = 1 $ '' DISK SECTOR BASE - EVENT ''                                      7
         ELSE $                 '' REPORT ''                                      8
         BCRECD = 512 $         '' BYTE COUNT - REPORT ''                         9
         J = 2400 $             '' DISK SECTOR BASE - REPORT ''                  10
         K = NOREPT $  '' NUMBER OF REPORT RECORDS ''                            11
      END IF $                                                                   12
      I = 0 $                                                                    13
      DO UNTIL I GQ K $                                                          14
         DSKRW (BCRECD,J,DISK'BUF) $'' READ RECORD ''                            15
         M = 0 $                                                                 16
         DO UNTIL M GQ (PINDX - 1) $  '' EXECUTE POLOSH STRING ''                17
            IF PNTRY(SMS) EQ ESCHP $  '' OPERATOR ''                             18
               THEN $                                                            19
               M = M + 1 $                                                       20
               TL = PNTRY(SMS) $                                                 21
               IF TL EQ 6 $                                                      22
                  THEN $                                                         23
                  CALL IFOP $  '' IF ''                                          24
                  ELSE $                                                         25
               END IF $                                                          26
               IF TL EQ 7 $ CALL ANDOP $ '' AND ''                               27
               IF TL EQ 8 $ CALL OROP $ '' OR ''                                 29
               IF TL EQ 9 $ CALL NOTOP $ '' NOT ''                               31
               IF TL EQ 10 $ BBEQ(CTYP) $ '' EQ ''                               33
               IF TL EQ 11 $ BBNE(CTYP) $ '' NE ''                               35
               IF TL EQ 12 $ BBLE(CTYP) $ '' LE ''                               37
               IF TL EQ 13 $ BBGE(CTYP) $ '' GE ''                               39
               IF TL EQ 14 $ BBLT(CTYP) $ '' LT ''                               41
               IF TL EQ 15 $ BBGT(CTYP) $ '' GT ''                               43
               IF TL EQ 18 $ BBPL(CTYP) $ '' + ''                                45
               IF TL EQ 19 $ BBMI(CTYP) $ '' - ''                                47
```

```
                IF TL EQ 20 $ BRML(CTYP) $ '' * ''
                IF TL EQ 21 $ BRDI(CTYP) $ '' / ''
                M = M + 1 $
                ELSE $
                IF PNTRY(SMS) EQ ATCHR $  '' CONSTANT ''
                     THEN $
                     M = M + 1 $
                     TI = PNTRY(SMS) $
                     TC = RCVAL(STIS) $
                     PSHO(1,TC,0,16) $
                     CTYP = 1 $ '' INTEGER ''
                     ELSE $
                     IF PNTRY(SMS) EQ SLASH $'' CHAR STRING  ''
                          THEN $
                          M = M + 1 $
                          TI = PNTRY(SMS) $
                          TC = DCOL(STIS) $
                     TK = RICARD(STCS) $
                     PSHO(1,TK,0,8) $
                     TC = RDLEN(STIS) $
                     PSHO(1,TC,0,8) $
                     PSHO(3,RSLASH,0,8) $
                          CTYP = 6 $ '' STRING ''
                          DIRECT $
                          .GLOBL    M.STRG
                          INC       M.STRG        ; STRING MODE
                          JOVIAL
                          ELSE $ '' COMPOOL ITEM POINTER ''
                     TK = TTYP(STIS) $
                     TL = TADDR(STIS) $
                     TM = TFB(STIS) $
                     TN = TNB(STIS) $
                     PSHO(TK,TL,TM,TN) $
                          CTYP = TTYP(STIS) $
                     END IF $
                END IF $
                M = M + 1 $        '' INCREMENT POLISH POINTER ''
           END IF $
      END DO $
      POPO(0,@TI,0,16) $ '' RESULT OF POLISH CODE ''
      IF TI NQ 0 $ '' IF TI = 1 PERFORM DESIRED TASK ''
           THEN $
           IF INSRC EQ 0 $
                THEN $
                EXEC RCV45 $
                DTRES(SOS) = 1$'' TAG NOT CHECKED FOR 11/45 ''
                ELSE $
           END IF $
           IF INSRC EQ 1 $
                THEN $
                EXEC RCVL1 $
                WAIT $
                ELSE $
           END IF $
           IF INSRC EQ 2 $
```

```
JOVIAL ISS                      *   HARRIS JOVIAL COMPILER
                                    ---VERSION -3A              PAGE    4
                 THEN $
                 EXEC RCVL2 $
                 WAIT $
                 ELSE $
             END IF $
             IF DTRES($INSRCS) EQ 0 $ '' THEN INVALID ACCESS ''
                 THEN $
                 '' SKIP TO TEST NEXT RECORD ''
                 ELSE $
                 IF MVTYP EQ 3 $ CALL DLETE $ '' DELETE ''
                 IF MVTYP EQ 4 $ CALL ADDRC $ '' ADD ''
                 IF MVTYP EQ 5 $ CALL CHNGE $ '' CHANGE ''
             END IF $
             ELSE $
         END IF $
         I = I + 1 $ '' INCREMENT RECORD COUNT ''
         J = J + 1 $ '' INCREMENT RECORD NUMBER ''
     END DO $
     RETURN $
      END SUBROUTINE $
 '' THESE SUBROUTINES NOT IMPLEMENTED AT THIS TIME ''
     ADDRC SUBROUTINE $
     END SUBROUTINE $
     DLETE SUBROUTINE $
     END SUBROUTINE $
     CHNGE SUBROUTINE $
     END SUBROUTINE $

     IFOP SUBROUTINE $
                     POPO(0,@TI,0,16) $
                     IF TI EQ 0 $
                         THEN $
                         PSHO(0,@TI,0,16) $
                         ELSE $
                         PSHO(0,@TI,0,16) $
                     END IF $
     END SUBROUTINE $

     ANDOP SUBROUTINE $
                     POPO(0,@TI,0,16) $
                     POPO(0,@TJ,0,16) $
                     IF TI EQ 1 $
                         THEN $
                         IF TJ EQ 1 $
                             THEN $
                             PSHO(0,@TJ,0,16) $
                             ELSE $
                             TI = 0 $
                             PSHO(0,@TI,0,16) $
                         END IF $
                         ELSE $
                         PSHO(0,@TI,0,16) $
                     END IF $
     END SUBROUTINE $
```

```
    ORDP SUBROUTINE  $
                              POPO(0,@TI,0,16) $
                              POPO(0,@TJ,0,16) $
                              IF TI EQ 1 $
                                 THEN $
                                 PSHO(0,@TI,0,16) $
                                 ELSE $
                                 PSHO(0,@TJ,0,16) $
                              END IF $
    END SUBROUTINE $

    NOTOP SUBROUTINE $
                              POPO(0,@TI,0,16) $
                              IF TI EQ 1 $
                                 THEN $
                                 TI = 0 $
                                 ELSE $
                                 TI = 1 $
                              END IF $
                              PSHO(0,TI,0,16) $
    END SUBROUTINE $
    DIRECT
;------------------------------------------------------------------------------
        .GLOBL  BREQ            ; EQUAL COMPARISON
;------------------------------------------------------------------------------
        .GLOBL  M.STRG
BREQ:
        MOV     R0,-(SP)        ; SAVE REGISTERS
        MOV     R1,-(SP)
        MOV     R2,-(SP)
        MOV     R3,-(SP)
        MOV     R4,-(SP)
        LDF     (MSTK)+,AC0     ; CONVERT TO FIXED
        STCFI   AC0,-(MSTK)     ;
        TST     (MSTK)+         ; SKIP 1ST WORD
        MOV     (MSTK)+,R0      ; GET CTYP, TYPE OF DATA
        ASL     R0              ; MULTIPLY BY 2
        JMP     @TEER(R0)       ; BRANCH ACCORDING TO TYPE
TEER:   .WORD   LOGCL
        .WORD   INTGR
        .WORD   FLTNG
        .WORD   CHARS
        .WORD   CHARS
        .WORD   CHARS
        .WORD   SCHAR
LOGCL:
INTGR:
FLTNG:
CHARS:
        IFEQ    TRU             ; BRANCH IF EQUAL
        CLR     -(MSTK)         ; PUSH 0 ON MSTK, FALSE
        BR      OUTT            ;
TRU:    MOV     #1,-(MSTK)      ; PUSH 1 ON MSTK, TRUE
OUTT:   CLR     -(MSTK)         ; PUSH LSB OF 0
        JMP     RETN            ; RESTORE REG.
```

```
JOVIAL ISS                              *    HARRIS JOVIAL COMPILER
                                             ---VERSION -3A            PAGE    6
SCHAR:                                  ; CHAR STRING                              177
        TST     M.STRG                  ; STRING MODE                              177
        BEQ     CHARS                                                              177
        CLR     R0                                                                 177
        MOV     #TMPA,R1                                                           177
1$:     CMP     (MSTK),#57              ; SLASH INDICATES DATA STRING              177
        BNE     3$                                                                 177
        CMP     (MSTK)+,(MSTK)+                                                    177
        MOV     (MSTK)+,R2              ; DOWN - LENGTH OF STRING                  177
        TST     (MSTK)+                                                            177
        MOV     (MSTK)+,R3              ; ADDR OF START COLUMN IN ICARD            177
        CMP     R2,#30.                 ; MAX LENGTH OF 30                         177
        BLE     4$                                                                 177
        MOV     #30.,R2                                                            177
4$:     MOVB    (R3)+,(R1)+                                                        177
        SOB     R2,4$                                                              177
5$:     TST     R0                      ; LOADED BOTH STRINGS ?                    177
        BNE     10$                     ;                                          177
        INC     R0                                                                 177
        MOV     #TMPB,R1                                                           177
        BR      1$                                                                 177
3$:     CMP     (MSTK)+,(MSTK)+         ; STEP STACK 2  - COMPOOL ENTRY            177
        MOV     (MSTK)+,R2              ; PUT CHAR COUNT IN R2                     177
        CMP     R2,30.                  ; MAX LENGTH OF 30                         177
        BLE     6$                                                                 177
        MOV     #30.,R2                                                            177
6$:     TST     (MSTK)+                 ;                                          177
        MOV     (MSTK)+,R3              ; PUT ADDRESS IN R3                        177
        MOV     #30.,R4                                                            177
        SUB     R2,R4                                                              177
7$:     MOVB    (R3)+,(R1)+             ; TRANSFER DATA TO TEMP AREA               177
        SOB     R2,7$                   ;                                          177
8$:     MOVB    #40,(R1)+               ; FILL WITH BLANKS                         177
        SOB     R4,8$                   ;                                          177
        BR      5$                                                                 177
10$:    MOV     #30.,R0                                                            177
        MOV     #TMPA,R1                                                           177
        MOV     #TMPB,R2                                                           177
11$:    CMPB    (R1)+,(R2)+                                                        177
        BNE     15$                     ; BRANCH IF NOT EQUAL                      177
        SOB     R0,11$                                                             177
        MOV     #1,-(MSTK)              ; PUSH 1 ON STACK FOR EQ.                  177
        BR      12$                                                                177
15$:    CLR     -(MSTK)                 ; PUSH 0 ON STACK FOR NE                   177
12$:    CLR     -(MSTK)                 ; PUSH LSB                                 177
RETN:                                                                              177
        MOV     (SP)+,R4                ; RESTORE REGISTERS                        177
        MOV     (SP)+,R3                ;                                          177
        MOV     (SP)+,R2                ;                                          177
        MOV     (SP)+,R1                ;                                          177
        MOV     (SP)+,R0                ;                                          177
        RTS     PC                      ; RETURN                                   177
TMPA:   .BLKB   30.                     ; TEMP LOCATIONS A                         177
TMPB:   .BLKB   30.                     ; TEMP LOCATIONS B                         177
;--------------------------------------------------------------------------        177
```

```
JOVIAL ISS                               *   HARRIS JOVIAL COMPILER        PAGE    7
                                             ---VERSION -3A                              177
        .GLOBL  BBNE                                                                     177
;----------------------------------------------------------------------                  177
BBNE:                                                                                    177
        JSR     PC,BBEQ                                                                  177
        TST     (MSTK)+                                                                  177
        TST     (MSTK)+                                                                  177
        BEQ     2$                                                                       177
        MOV     #0,-(MSTK)                                                               177
        BR      3$                                                                       177
2$:     MOV     #1,-(MSTK)                                                               177
3$:                                                                                      177
        MOV     #0,-(MSTK)                                                               177
        RTS     PC                                                                       177
;----------------------------------------------------------------------                  177
        .GLOBL  BBLE                                                                     177
;----------------------------------------------------------------------                  177
BBLE:                                                                                    177
        MOV     R0,-(SP)                                                                 177
        MOV     #2,R0                                                                    177
        BR      BCMP                                                                     177
;----------------------------------------------------------------------                  177
        .GLOBL  BBGE                                                                     177
;----------------------------------------------------------------------                  177
BBGE:                                                                                    177
        MOV     R0,-(SP)                                                                 177
        MOV     #4,R0                                                                    177
        BR      BCMP                                                                     177
;----------------------------------------------------------------------                  177
        .GLOBL  BBLT                                                                     177
;----------------------------------------------------------------------                  177
BBLT:                                                                                    177
        MOV     R0,-(SP)                                                                 177
        MOV     #6,R0                                                                    177
        BR      BCMP                                                                     177
;----------------------------------------------------------------------                  177
        .GLOBL  BBGT                                                                     177
;----------------------------------------------------------------------                  177
BBGT:                                                                                    177
        MOV     R0,-(SP)                                                                 177
        MOV     #10,R0                                                                   177
BCMP:                                                                                    177
        LDF     (MSTK)+,AC0                                                              177
        STCFI   AC0,-(MSTK)                                                              177
        TST     (MSTK)+                                                                  177
        CMP     #3,(MSTK)                                                                177
        BLT     1$                                                                       177
        JMP     SERR1                   ; SYNTAX ERROR, BAD TYPE                         177
1$:     CMP     #2,(MSTK)+                                                               177
        BNE     2$                                                                       177
        INC     M.FLT                                                                    177
2$:                                                                                      177
        JMP     BCMP0(R0)                                                                177
BCMP0:  .WORD   OLE                                                                      177
        .WORD   OGE                                                                      177
        .WORD   OLT
```

```
        .WORD   OGT                                                                   177
OLE:    IFLO    TRU                                                                   177
        BR      FLS                                                                   177
OGE:    IFGO    TRU                                                                   177
        BR      FLS                                                                   177
OLT:    IFLS    TRU                                                                   177
        BR      FLS                                                                   177
OGT:    IFGR    TRU                                                                   177
FLS:    MOV     #0,-(MSTK)                                                            177
        MOV     #0,-(MSTK)                                                            177
        MOV     (SP)+,R0                                                              177
        RTS     PC                                                                    177
                                                                                      177
;-------------------------------------------------------------------------------------  177
                                  ; ERROR RETURNS                                     177
SERR1:  MOV     (SP)+,R0          ; RESTORE STACK                                     177
        CMP     (MSTK)+,(MSTK)+   ; POP OPERANDS OFF STACK AND SKIP                   177
                                  ; GENERATE ERROR MESSAGE                            177
        NOP                                                                           177
        RTS     PC                ; RETURN                                            177
;-------------------------------------------------------------------------------------  177
                                                                                      177
;------------------------------------------------------------------------------        177
        .GLOBL  BBPL                                                                  177
;------------------------------------------------------------------------------        177
BBPL:                                                                                 177
        MOV     R0,-(SP)                                                              177
        MOV     #2,R0                                                                 177
        BR      BBOP                                                                  177
;------------------------------------------------------------------------------        177
        .GLOBL  BBMI                                                                  177
;------------------------------------------------------------------------------        177
BBMI:                                                                                 177
        MOV     R0,-(SP)                                                              177
        MOV     #4,R0                                                                 177
        BR      BBOP                                                                  177
;------------------------------------------------------------------------------        177
        .GLOBL  BBML                                                                  177
;------------------------------------------------------------------------------        177
BBML:                                                                                 177
        MOV     R0,-(SP)                                                              177
        MOV     #6,R0                                                                 177
        BR      BBOP                                                                  177
;------------------------------------------------------------------------------        177
        .GLOBL  BBDI                                                                  177
;------------------------------------------------------------------------------        177
BBDI:                                                                                 177
        MOV     R0,-(SP)                                                              177
        MOV     #10,R0                                                                177
BBOP:                                                                                 177
        LDF     (MSTK)+,AC0                                                           177
        STCFI   AC0,-(MSTK)                                                           177
        TST     (MSTK)+                                                               177
        CMP     #3,(MSTK)                                                             177
        BNE     1$                                                                    177
```

```
JOVIAL ISS                              *    HARRIS JOVIAL COMPILER
                                             ---VERSION -3A              PAGE    9
          JMP      SERR1             ; SYNTAX ERROR - BAD TYPE                          177
1$:                                                                                     177
          CMP      #2,(MSTK)+                                                           177
          BNE      2$                                                                   177
          INC      M.FLT                                                                177
2$:                                                                                     177
          JMP      BROPO(R0)                                                            177
BROPO:    .WORD    OPL                                                                  177
          .WORD    OMI                                                                  177
          .WORD    OML                                                                  177
          .WORD    ODI                                                                  177
OPL:      RADD                                                                          177
          BR       OND                                                                  177
OMI:      RSUB                                                                          177
          BR       OND                                                                  177
OML:      RMUL                                                                          177
          BR       OND                                                                  177
ODI:      RDIV                                                                          177
OND:      MOV      (SP)+,R0                                                             177
          RTS      PC                                                                   177
;-------------------------------------------------------------------------------       177
;     SUBROUTINE TO POP DATA FROM MSTK TO A COMPOOL ITEM                                177
;-------------------------------------------------------------------------------       177
          .GLOBL   POPO              ; CALL = POPO (MODE,BASE,SB,NB)                    177
POPO:                                                                                   177
          POPB     0,0,0,0,PONB,0,0,8                                                   177
          POPB     0,0,0,0,POSB,0,0,8                                                   177
          POPB     0,0,0,0,POBASE,0,0,16                                                177
          POPB     0,0,0,0,POMODE,0,0,8                                                 177
          JSR      R4,M.POPB                                                            177
          .GLOBL   M.POPB                                                               177
POMODE:   .BYTE    0                                                                    177
          .BYTE    0                                                                    177
POBASE:   .WORD    0                                                                    177
POSB:     .BYTE    0                                                                    177
PONB:     .BYTE    0                                                                    177
          RTS      PC                                                                   177
          .GLOBL   PSHO                                                                 177
;-------------------------------------------------------------------------------       177
;     SUBROUTINE TO PUSH A COMPOOL ITEM ONTO MSTK                                       177
;-------------------------------------------------------------------------------       177
PSHO:                                ; CALL = PSHO(MODE,BASE,SB,NB)                     177
          POPB     0,0,0,0,PUNB,0,0,8                                                   177
          POPB     0,0,0,0,PUSB,0,0,8                                                   177
          POPB     0,0,0,0,PUBASE,0,0,16                                                177
          POPB     0,0,0,0,PUMODE,0,0,8                                                 177
          JSR      R4,M.PUSH                                                            177
PUMODE:   .BYTE    0                                                                    177
          .BYTE    0                                                                    177
PUBASE:   .WORD    0                                                                    177
PUSB:     .BYTE    0                                                                    177
PUNB:     .BYTE    0                                                                    177
          RTS      PC                                                                   177
          .GLOBL   PSHOB                                                                177
PSHOB:                                                                                  177
```

```
JOVIAL ISS                              *   HARRIS JOVIAL COMPILER
                                            ---VERSION -3A            PAGE   10
           POPR     0,0,0,0,DBTYP,0,0,8                                           177
           POPR     0,0,0,0,DBADR,0,0,16                                          177
           POPR     0,0,0,0,DBFB,0,0,8                                            177
           POPR     0,0,0,0,DBNB,0,0,8                                            177
           JSR      R4,M.PUSH                                                     177
DBTYP:     .BYTE    0                                                             177
           .BYTE    0                                                             177
DBADR:     .WORD    0                                                             177
DBFB:      .BYTE    0                                                             177
DBNB:      .BYTE    0                                                             177
           RTS      PC                                                            177
         JOVIAL                                                                   177
         END PROGRAM $                                                            177
```

DATA BASE REFERENCE LISTING

S	5	13	0	0
NENT	0	3	0	0
ECARD	0	14	0	8
ESCHR	74	9	0	8
SLASH	74	9	8	6
ATCHR	75	9	0	8
RINDX	566	9	0	16
MVTYP	576	9	0	16
MSTYP	577	9	0	16
NOEVNT	601	9	0	16
NOREPT	602	9	0	16
BCRECD	603	9	0	16
TI	604	9	0	16
TJ	605	9	0	16
TC	606	9	0	16
TK	607	9	0	16
TL	610	9	0	16
TM	611	9	0	16
TN	612	9	0	16
CTYP	616	9	0	16
INSRC	622	9	0	16
PNTRY	740	14	0	8
TTYP	1060	9	0	16
TER	1104	9	0	16
TNH	1130	9	0	16
TADDR	1154	9	0	16
CVAL	1060	10	0	16
DCOL	1060	9	0	8
DLEN	1060	9	8	8
DTRES	1661	9	0	16
DISK BUF	5777	2	1	256
EVENT	6501	2	1	60
I	1	13	0	16
J	2	13	0	16
K	3	13	0	16
M	5	13	0	16

COMPOOL SPACE USED = 1133

0 ERRORS

```
$RUN OPT
$FINISH
TIME:-00:14:19
```

```
$JOB COMPILE RCVIP
DATE:-22-AUG-77
TIME:-00:29:27
$RUN PIP
PIP V10-03A
*RPN.SRC/DE

*COMPOL.001/DE

*COMPOL.001<COMPOL.ISS

*RPN.SRC<BI:/FA

$EOD

*
$RUN JOVIAL
```

```
JOVIAL ISS                                  *   HARRIS JOVIAL COMPILER
                                                ---VERSION -3A          PAGE    1
      'PROGRAM RCV45 $                                                                  1
''------------------------------------------------------------------------''            2
'' RCVIP - RCV45 IS VERSION OF RCVIP TO RESIDE IN 11/45. IT IS THE        ''            2
'' SAME AS RCVL1 EXCEPT THAT IT DOES NOT CHECK THE TAG, IT ONLY           ''            2
'' EXECUTES THE COMMAND AND RETURNS TO CCXEC                              ''            2
''------------------------------------------------------------------------''            2
      I = 0 $                                                                           2
           IF MSTYP EQ 1 $ '' SIGNIFIES EVENT ''                                        3
                THEN $                                                                  4
                IF MVTYP EQ 1 $                                                         5
                     THEN $                                                             6
                     '' DISPLAY NOT AVAILABLE PRESENTLY ''                              7
                     ELSE $                                                             7
                     IF MVTYP EQ 2 $ WRTKW(@EVENT,80) $ ''PRINT ''                      8
                END IF $                                                                10
                '' IF MVTYP GR 2 THEN RETURN TO 11/45 ''                                11
                ELSE $                                                                  11
                IF MSTYP EQ 2 $ '' REPORT ''                                            12
                     THEN $                                                             13
                     '' CODE TO HANDLE REPORT WILL GO HERE ''                           14
                     ELSE $                                                             14
                END IF $                                                                15
           END IF $                                                                     16
      DIRECT $                                                                          17
         NOTIFY  #INP45                                                                 17
         EXIT                                                                           17
      JOVIAL                                                                            17
      END PROGRAM $                                                                     17
```

```
   DATA BASE REFERENCE LISTING
MVTYP                          576     9     0    16
MSTYP                          577     9     0    16
EVENT                         6501     2     1    60
I                                1    13     0    16
COMPOOL SPACE USED =   1093

  0 ERRORS
```

SFINISH
TIME:-00:31:27

```
$JOB COMPILE RCVIP
DATE:-22-AUG-77
TIME:-00:32:58
$RUN PIP
PIP V10-03A
*RPN.SRC/DE

*COMPOL.001/DE

*COMPOL.001<COMPOL.ISS

*RPN.SRC<RI:/FA

$EOD

*
$RUN JOVIAL
```

```
JOVIAL ISS                                 *   HARRIS JOVIAL COMPILER
                                               ---VERSION -3A            PAGE    1
    'PROGRAM RCVL1 $                                                                1
''-------------------------------------------------------------------''             2
'' RCVIP - RECEIVES INPUT FROM THE 11/45 IN RESPONSE TO A QUERY      ''             2
'' AND CHECKS TAG AND THEN EXECUTES THE COMMAND ''                                  2
    ''NOW RECORD OR DATA IS IN COMPOOL WAITING TO BE OUTPUT''                       2
    ''NEED TO CHECK TAG BEFORE OUTPUT''                                             2
    ''DETAG IS DIFFERENT FOR EVENT AND REPORT ; MSTYP IS QUALIFIER''                2
''-------------------------------------------------------------------''             2
    I = 0 $                                                                         2
    CALL DETAG  $                                                                   3
    IF DTRES ($I$) EQ 0 $  ''THEN TAG NO GOOD''                                     4
        THEN $                                                                      5
        SECERR($I$) = SECERR($I$) + 1 $                                             6
        ''IF TOO MANY SECURITY VIOLATIONS - LOCK UP TERMINAL''                      7
        IF SECERR($I$) GR ANSV $                                                    7
            THEN $                                                                  8
            LRDY($I$) = 2 $                                                         9
            ELSE $                                                                  10
        END IF $                                                                    11
        ''IF TOO MANY, LRDY IS SET=2 - TERMINAL LOCKS''                             12
        ELSE $  ''TAG IS OK''                                                       12
        IF MSTYP EQ 1 $ '' SIGNIFIES EVENT ''                                       13
            THEN $                                                                  14
            IF MVTYP EQ 1 $                                                         15
                THEN $                                                              16
                '' DISPLAY NOT AVAILABLE PRESENTLY ''                               17
                ELSE $                                                              17
                IF MVTYP EQ 2 $ WRTKW(@EVENT,80) $ ''PRINT ''                       18
            END IF $                                                                20
            '' IF MVTYP GR 2 THEN RETURN TO 11/45 ''                                21
            ELSE $                                                                  21
            IF MSTYP EQ 2 $ '' REPORT ''                                            22
                THEN $                                                              23
                '' CODE TO HANDLE REPORT WILL GO HERE ''                            24
                ELSE $                                                              24
            END IF $                                                                25
        END IF $                                                                    26
    END IF $                                                                        27
    DIRECT $                                                                        28
       NOTIFY  #INP45                                                               28
       EXIT                                                                         28
    JOVIAL                                                                          28
    END PROGRAM $                                                                   28
```

```
    DATA BASE REFERENCE LISTING
S                                5    13    0    0
MVTYP                          576     9    0   16
MSTYP                          577     9    0   16
ANSV                           624     9    0   16
LRDY                          1653     9    0   16
DTRES                         1661     9    0   16
SECERR                        1664     9    0   16
EVENT                         6501     2    1   80
I                                1    13    0   16
COMPOOL SPACE USED =  1093

   0 ERRORS
```

$FINISH
TIMF:-00:35:37

```
$JOB COMPILE ACCRM
DATE:-22-AUG-77
TIME:-00:43:07
$RUN PIP
PIP V10-03A
*RPN.SRC/DE

*COMPOL.001/DE

*COMPOL.001<COMPOL.ISS

*RPN.SRC<BI:/FA

$EOD

*
$RUN JOVIAL
```

```
JOVIAL ISS                                  *   HARRIS JOVIAL COMPILER
                                                ---VERSION -3A          PAGE    1
      'PROGRAM ACCPM                                                                  1
  ''-----------------------------------------------------------------------''        1
  ''    ACCPM - ACCESS CONTROL CENTER PM - ACCEPTS INPUT FROM THE         ''        1
  ''         LA36 KEYBOARD IN ORDER TO SET UP ACCESS PRIORITIES FOR       ''        1
  ''         EACH LSI TERMINAL.  THEN IT TURNS ON THE INPUT PROCESSORS    ''        1
  ''         FOR THE LSI'S AND FOR THE 11/45 AND STARTS THE SYSTEM        ''        1
  ''         CYCLING.                                                     ''        1
  ''    INPUTS:                                                           ''        1
  ''         KBUF = KEYBOARD INPUT BUFFER                                 ''        1
  ''                                                                      ''        1
  ''    OUTPUTS:                                                          ''        1
  ''         ACKT  = ACCESS KEYWORD TABLE - 1 ENTRY/LSI TERMINAL          ''        1
  ''         RNTBL = RANDOM NUMBER TABLE FOR USE BY TAG/DETAG             ''        1
  ''                                                                      ''        1
  ''-----------------------------------------------------------------------''        1
      I = 0 $                                                                         1
AGIN.                                                                                 2
      LREAD (@TM,1) $                                                                 2
      WRTKW ( TM   ,62)  $  ''PROMPT INPUT''                                          3
A1.   REDKW (@KBUF,@TI) $  ''READ TERMINAL NUMBER''                                   4
      IF TI GR 2 $                                                                    5
           THEN $                                                                     6
           LREAD (@TM,4) $                                                            7
           WRTKW ( TM   ,24) $ ''ERROR ; PLEASE REENTER''                             8
           GOTO A1 $                                                                  9
           ELSE $                                                                     10
      END IF $                                                                        11
      J = 0 $                                                                         12
      IF IKBUF($0$) EQ A1 $ J = 1  $  ''SET TABLE INDEX''                             13
      IF IKBUF($0$) EQ A2 $ J = 2  $                                                  15
      IF J EQ 0 $                                                                     17
           THEN $                                                                     18
           LREAD (@TM,4) $                                                            19
           WRTKW ( TM   ,24) $ ''ERROR ; PLEASE REENTER''                             20
           GOTO A1 $  ''ERROR ; PLEASE REENTER''                                      21
           ELSE $                                                                     22
      END IF $                                                                        23
      LREAD (@TM,2) $                                                                 24
      WRTKW ( TM   ,48) $  ''PROMPT INPUT''                                           25
A2.   REDKW (@KBUF,@TI) $  ''READ PRIORITY NUMBER''                                   26
      IF TI GR 2 $                                                                    27
           THEN $                                                                     28
           LREAD (@TM,4) $                                                            29
           WRTKW ( TM   ,24) $ ''ERROR ; PLEASE REENTER''                             30
           GOTO A2  $                                                                 31
           ELSE $                                                                     32
      END IF $                                                                        33
      IF IKBUF($0$) GR 3 $                                                            34
           THEN $                                                                     35
           LREAD (@TM,4) $                                                            36
           WRTKW ( TM   ,24) $ ''ERROR ; PLEASE REENTER''                             37
           GOTO A2 $                                                                  38
           ELSE $                                                                     39
      END IF $                                                                        40
      CASE IKBUF($0$)  $                                                              41
```

```
          OF 0 $                                                                              42
          ACKT($J$) = UACC   $   ''UNCLASSIFIED ACCESS''                                      43
          OF 1 $                                                                              44
          ACKT($J$) = CACC   $   ''CONFIDENTIAL ACCESS''                                      45
          OF 2 $                                                                              46
          ACKT($2$) = SACC   $   ''SECRET ACCESS''                                            47
          OF 3 $                                                                              48
          ACKT($3$) = TSACC $   ''TOP SECRET ACCESS''                                         49
     END CASE $                                                                               50
     LREAD (@TM,3) $                                                                          51
     WRTKW ( TM  ,128)  $  ''PROMPT FOR NEXT TERMINAL OR START''                              52
     REDKW (@KBUF,@TI)  $  ''READ INPUT''                                                     53
     IF TI LQ 1 $ GOTO AGIN $  ''ENTER ANOTHER TERMINAL''                                     54
     ''ELSE A     -BEGIN-     WAS ENTERED''                                                   56
     ''PERFORM SOME INITIALIZATION BEFORE STARTING SYSTEM''                                   56
     ''THE TRANSLATION TABLE AND SEED NUMBER ARE INITIALIZED''                                56
     ''IN THE COMPOOL.  NOW CREATE A 256 WORD RANDOM''                                        56
     ''NUMBER TABLE ''                                                                        56
     RNGIN  (@RNTBL) $                                                                        56
     ''NOW TURNON INPUT PROCESSORS IN LSI S''                                                 57
     IF ACKT($1$) NQ 0 $                                                                      57
          THEN $                                                                              58
          DIRECT $                                                                            59
          TURNON   #IPLS1                                                                     59
          JOVIAL                                                                              59
          ELSE $                                                                              59
     END IF $                                                                                 60
     IF ACKT($2$) NQ 0 $                                                                      61
          THEN $                                                                              62
          DIRECT $                                                                            63
          TURNON   #IPLS2                                                                     63
          JOVIAL                                                                              63
          ELSE $                                                                              63
     END IF $                                                                                 64
     ''TURNON INPUT PROCESSOR FOR 11/45''                                                     65
     DIRECT $                                                                                 65
        TURNON   #IPL45                                                                       65
 ; TURNON INPUT POLLING IN 11/45                                                              65
        TURNON   #INP45                                                                       65
        EXIT                                                                                  65
        MSTK = %5                                                                             65
        .GLOBL   RNGIN                                                                        65
        .GLOBL   RANDU                                                                        65
RNGIN:                                                                                        65
        MOV      R0,-(SP)          ;SAVE REGISTERS                                            65
        MOV      R1,-(SP)          ;                                                          65
        MOV      #256.,R0          ;LENGTH OF RN TABLE                                        65
        LDF      (MSTK)+,AC0       ;CONVERT ADDR OF RNTBL                                     65
        STCFI    AC0,-(MSTK)       ;  TO FIXED                                                65
        TST      (MSTK)+           ;                                                          65
        MOV      (MSTK)+,R1        ; PUT ADDR IN R1                                           65
        MOV      R5,-(SP)          ;SAVE MSTK PTR                                             65
1$:     JSR      R5,RANDU          ;                                                          65
        BR       2$                ;                                                          65
        .WORD    I1                ;                                                          65
```

```
        .WORD   I2                  ;                                              65
        .WORD   X                   ;                                              65
2$:                                                                                65
        MOV     X,(R1)+             ;PUT RANDOM NUMBER IN TABLE                    65
        SOB     R0,1$               ;255 TIMES                                     65
        MOV     (SP)+,R5            ;RESTORE MSTK                                  65
        MOV     (SP)+,R1            ;  AND OTHER REG.                              65
        MOV     (SP),R0             ;                                              65
        RTS     PC                                                                 65
I1:     .WORD   0                                                                  65
I2:     .WORD   0                                                                  65
X:      .WORD   0                                                                  65
                                                                                   65
        .GLOBL  LREAD                                                              65
LREAD:                              ;THIS ROUTINE STORES THE ADDR OF               65
                                    ; NON-COMPOL ITEMS(PRINT BUFFERS) INTO         65
                                    ; A COMPOL ITEM FOR US BY A SUBROUTINE         65
        LDF     (MSTK)+,AC0         ; CONVERT PARAMETER TO INTEGER                 65
        STCFI   AC0,-(MSTK)         ;                                              65
        MOV     R0,-(SP)            ; SAVE REG                                     65
        MOV     R1,-(SP)                                                           65
        TST     (MSTK)+                                                            65
        MOV     (MSTK)+,R0          ; MSG #                                        65
        LDF     (MSTK)+,AC0         ; CONVERT PARAMETER                            65
        STCFI   AC0,(MSTK)+         ;                                              65
        TST     (MSTK)+                                                            65
        ASL     R0                  ; CALCULATE OFFSET FROM MSG #                  65
        MOV     #BAT,R1             ; ADDR TABLE                                   65
        ADD     R0,R1               ;                                              65
        MOV     @R1,@(MSTK)+        ; MOVE BUFF ADDR TO COMPOL ITEM                65
        MOV     (SP)+,R1            ; RESTORE REG                                  65
        MOV     (SP)+,R0            ;                                              65
        RTS     PC                                                                 65
BAT:    .WORD   0                   ; BUFFER ADDR TABLE                            65
        .WORD   MSG1                                                               65
        .WORD   MSG2                                                               65
        .WORD   MSG3                                                               65
        .WORD   MSG4                                                               65
; I/O OUTPUT BUFFERS ----                                                          65
MSG1:   .ASCII  / ACCESS CONTROL CENTER READY/                                     65
        .BYTE   12,15                                                              65
        .ASCII  / PLEASE ENTER TERMINAL NUMBER/                                    65
        .BYTE   12,15                                                              65
        .EVEN                                                                      65
MSG2:   .ASCII  / ENTER PRIORITY OF 0-3 FOR DESIGNATED TERMINAL/                   65
        .BYTE   12,15                                                              65
        .EVEN                                                                      65
MSG3:   .ASCII  / IF ALL TERMINALS HAVE BEEN ENTERED/                              65
        .BYTE   12,15                                                              65
        .ASCII  / ENTER 'BEGIN' TO ENABLE QUERY PROCESSING/                        65
        .BYTE   12,15                                                              65
        .ASCII  / ELSE PRESS 'RETURN' TO ENTER ANOTHER TERMINAL/                   65
        .BYTE   12,15                                                              65
        .EVEN                                                                      65
MSG4:   .ASCII  / ERROR; PLEASE REENTER/                                           65
```

```
        .BYTE   12,15                                               65
        .EVEN                                                       65
      JOVIAL                                                        65
      END PROGRAM $                                                 65
```

```
    DATA BASE REFERENCE LISTING
S                          5    13    0     0
A1                        67    14    8     8
A2                        70    14    0     8
TI                       604     9    0    16
TM                       611     9    0    16
UACC                     625     9    0    16
SACC                     626     9    0    16
CACC                     627     9    0    16
TSACC                    630     9    0    16
RNTBL                   1200     2    1   256
KBUF                    1600     2    1    40
IKBUF                   1600    14    0     8
ACKT                    1650     9    0    16
J                          2    13    0    16
COMPOOL SPACE USED =  1098

   0 ERRORS
```

SFINISH
TIME:-00:47:48

```
$JOB COMPILE IPLST
DATE:-22-AUG-77
TIME:-00:51:46
$RUN PIP
PIP V10-03A
*RPN.SRC/DE

*COMPOL.001/DE

*COMPOL.001<COMPOL.ISS

*RPN.SRC<BI:/FA

$EOD

*
$RUN JOVIAL
```

```
JOVIAL ISS                             *    HARRIS JOVIAL COMPILER
                                            ---VERSION -3A           PAGE    1
      'PROGRAM IPLSI $                                                                  1
 ''-----------------------------------------------------------------------''          2
 ''    IPLSI - INPUT PROCESSING FOR THE LSI-11'S ; THIS ROUTINE           ''          2
 ''       ACCEPTS COMMAND INPUTS FROM THE KEYBOARD AND INITIATES          ''          2
 ''       THE CORRECT PROCESSING                                          ''          2
 ''    INPUTS:                                                            ''          2
 ''          LRDY = TABLE WITH HANDSHAKE FLAGS TO 11-45                   ''          2
 ''    OUTPUTS:                                                           ''          2
 ''          CBUF = TABLE OF COMMAND INPUT BUFFERS                        ''          2
 ''          DBUF = TABLE OF DATA BUFFERS                                 ''          2
 ''-----------------------------------------------------------------------''          2
      L = 0 $                                                                           2
      PDNDX (@TI) $'' PROCESSOR DEPENDENT INDEX RETURNED IN TI ''                       3
      I = TI $    '' PUT INDEX IN I BECAUSE I NOT PART OF COMPOOL ''                    4
      DO WHILE GO EQ 0 $                                                                5
           J = @BYTEC(SIS) $                                                            6
           K = @CBUF(SIS) $                                                             7
           REDKW(J,K) $ ''READ KEYBOARD''                                               8
           ''EVENTUALLY - CODE TO ACCEPT DATA FOR ADD OR CHANGE''                       9
           ''WOULD GO HERE''                                                            9
           LRDY(SIS) = 1 $  ''NOTIFY 45 THAT COMMAND IS READY''                         9
           DO UNTIL LRDY (SIS) EQ 0 $ ''WAIT UNTIL 45 FINISHED''                       10
           L = 0 $                                                                     11
           END DO $                                                                    12
      END DO $                                                                         14
      DIRECT $                                                                         15
        EXIT                                                                           15
 ; PDNDX CALCULATES A UNIQUE PROCESSOR INDEX USING SYSTEM VARIABLE                     15
 ; SY.K1 AND PUTS IT IN TI, A COMPOOL ITEM TO RETURN TO JOV.                           15
        .GLOBL  PDNDX                                                                  15
        .GLOBL  SY.K1                                                                  15
        MSTK = %5                                                                      15
PDNDX :                                                                                15
         LDF      (MSTK)+,AC0        : CONVERT ADDR OF TI                              15
         STCFI    AC0,-(MSTK)        :                                                 15
         TST      (MSTK)+            :                                                 15
         TST      #SY.K1             : TEST SYSTEM VARIABLE                            15
         BNE      1$                 :                                                 15
         CLR      @(MSTK)+           : IF 0, CLEAR TI                                  15
         BR       NND                :                                                 15
1$:                                                                                    15
         CMP      #SY.K1,#2          : IF 2                                            15
         BNE      2$                 :                                                 15
         MOV      #1,@(MSTK)+        : PUT INDEX OF 1 IN TI                            15
         BR       NND                :                                                 15
2$:                                                                                    15
         CMP      #SY.K1,#4          : IF 4                                            15
         BNE      NND                :                                                 15
         MOV      #2,@(MSTK)+        : PUT INDEX OF 2 IN TI                            15
         NND:                                                                          15
         RTS      PC                 ; RETURN                                          15
      JOVIAL                                                                           15
      END PROGRAM $                                                                    15
```

```
DATA BASE REFERENCE LISTING
S                          5   13   0    0
TI                       604    9   0   16
GD                       623    9   0   16
LRDY                    1653    9   0   16
BYTEC                   1656    9   0   16
CBUF                    1667    9   0   16
I                          1   13   0   16
J                          2   13   0   16
K                          3   13   0   16
L                          4   13   0   16
COMPOOL SPACE USED =  1093

  0 ERRORS
```

$RUN OPT
$FINISH
TIME:-00:54:28

```
$JOB CCNTL COMPILE
DATE:-22-AUG-77
TIME:-00:57:07
$RUN PIP
PIP V10-03A
*RPN.SRC/DF

*COMPOL.001/DE

*COMPOL.001<COMPOL.ISS

*RPN.SRC<BI:/FA

$EOD

*
$RUN JOVIAL
```

JOVIAL TSS * HARRIS JOVIAL COMPILER

PAGE $ ---VERSION -3A PAGE 1

```
      CCNTL SUBROUTINE $                                                          2
  ''---------------------------------------------------------------------''       3
  ''   COMMAND PARSING CONTROL - THIS ROUTINE CALLS THE PROPER ROUTINES''         3
  ''      TO TRANSLATE AND EXECUTE A COMMAND LINE.                       ''       3
  ''   INPUTS ARE:                                                       ''       3
  ''      ICARD = TABLE CONTAINING THE COMMAND LINE                      ''       3
  ''       ICOL = POINTER TO ICARD                                       ''       3
  ''      NOCHR = NUMBER OF CHARACTERS IN ICARD                          ''       3
  ''   OUTPUTS ARE:                                                      ''       3
  ''      EXECUTION OF THE COMMAND, SETS FLAG PRDY TO SIGNAL IT IS       ''       3
  ''      READY FOR A NEW COMMAND.                                       ''       3
  ''---------------------------------------------------------------------''       3
  '' INITIALIZE TABLES AND POINTERS ''                                            3
      OPNDX = 0 $                                                                 3
      PINDX = 0 $                                                                 4
      J = 0 $                                                                     5
      DO UNTIL J EQ 40 $                                                          6
           OPPTR($J$) = 0 $                                                       7
           J = J + 1 $                                                            8
      END DO $                                                                    9
      J = 0 $                                                                     11
      DO UNTIL J EQ 80 $                                                          12
           PNTRY($J$) = 0 $                                                       13
           J = J + 1 $                                                            14
      END DO $                                                                    15
      ICOL = 0 $                                                                  17
  '' BEGIN LINE SCAN ''                                                           18
      DO UNTIL ICOL EQ NOCHR $                                                    18
      IF ICARD($ICOL$) NQ IB $ GOTO ARAND $                                       19
      NBLNK $                                                                     21
  '' NBLNK SEARCHES FOR THE FIRST NON-BLANK CHARACTER ''                          21
      DO UNTIL ICARD($ICOL$) NQ IB $                                              21
           ICOL = ICOL + 1 $                                                      22
      END DO $                                                                    23
ARAND.                                                                            25
           BLNK $                                                                 25
  '' BLNK SEARCHES FOR THE FIRST BLANK CHARACTER ''                               25
  ''      DOESN'T CHANGE ICOL, STORES COLUMN IN BPTR ''                           25
      BPTR = ICOL $                                                               25
      DO UNTIL ICARD($BPTR$) EQ IB $                                              26
           BPTR = BPTR + 1 $                                                      27
      END DO $                                                                    28
           ISLEN = BPTR - ICOL $                                                  30
           CALL KSCAN $                                                           31
           IF ICTYP EQ 0 $   '' MEANS NOT A KEYWORD ''                            32
                THEN $                                                            33
                CALL TTGEN     $   '' SEE IF COMPOOL ITEM ''                      34
                IF TTERR EQ 0 $    '' MEANS NOT A COMPOOL ITEM ''                 35
                     THEN $                                                       36
                     CALL COCON $  '' CHECK IF DECIMAL CONSTANT ''                37
                     IF DCR EQ 0 $ '' MEANS NOT DECIMAL CONSTANTS ''              38
                          THEN $                                                  39
                          DCOL($TTPTR$) = ICOL $                                  40
                          DLEN($TTPTR$) = ISLEN $                                 41
                          DCPTR = TTPTR $                                         42
```

```
                         TTPTR = TTPTR + 1 $
                         INTYP = 4 $  '' DENOTES CHAR STRING DATA ''
                         CALL POLGN $
                         ICOL = ICOL + ISLEN $
                         ELSE $
                         INTYP = 2 $  '' DENOTES DECIMAL CONSTANT ''
                         CALL POLGN $
                         ICOL = ICOL + ISLEN $
                    END IF $
                    ELSE $
                    INTYP = 0 $  '' COMPOOL ITEM ''
                    CALL POLGN $
                    ICOL = ICOL + ISLEN $
               END IF $
               ELSE $
               IF TKYNM EQ 23 $       MSTYP = 1 $ '' EVENT  ''
               IF TKYNM EQ 24 $       MSTYP = 2 $ '' REPORT''
               IF TKYNM EQ 1 $       MVTYP = 1 $  '' DISPLAY ''
               IF TKYNM EQ 2 $       MVTYP = 2 $  '' PRINT ''
               IF TKYNM EQ 3 $       MVTYP = 3 $  '' DELETE ''
               IF TKYNM EQ 4 $       MVTYP = 4 $  '' ADD ''
               IF TKYNM EQ 5 $       MVTYP = 5 $  '' CHANGE ''
               IF TKYNM GR 5 $
                    THEN $
                    IF TKYNM LS 23 $
                         THEN $
                         INTYP = 1 $ '' KEYWORD ''
                         CALL POLGN $
                         ELSE $
                    END IF $
                    ELSE $
               END IF $
               ICOL = ICOL + ISLEN $
          END IF $
     END DO $
     INTYP = 3 $  '' SIGNAL END OF INPUT TO POLSHGEN ''
     CALL POLGN $
'' INPUT LINE NOW TRANSLATED TO POLISH FORM, READY FOR CODE EXECUTION''
     CALL CCXEC $
     PRDY = 0 $  '' SIGNAL READY FOR NEXT COMMAND ''
     LRDY($INSRC$) = 0 $  '' SIGNAL INPUT PROCESSOR TO RESUME ''
     DIRECT $
        EXIT
     JOVIAL
     END SUBROUTINE $
     END PROGRAM $
```

```
  DATA BASE REFERENCE LISTING
S                                 5    13    0    0
ICARD                             0    14    0    8
IB                               50    14    0    8
ICOL                            106     9    0   16
ISLEN                           563     9    0   16
OPNDX                           564     9    0   16
ICTYP                           565     9    0   16
PINDX                           566     9    0   16
TKYNM                           567     9    0   16
TTPTR                           570     9    0   16
INTYP                           571     9    0   16
PRDY                            572     9    0   16
DCR                             573     9    0   16
DCPTR                           575     9    0   16
MVTYP                           576     9    0   16
MSTYP                           577     9    0   16
NOCHR                           600     9    0   16
TTEPR                           620     9    0   16
BPTR                            621     9    0   16
INSRC                           622     9    0   16
OPPTR                           670    14    0    8
PNTRY                           740    14    0    8
DCOL                           1060     9    0    8
DLEN                           1060     9    8    8
LRDY                           1653     9    0   16
J                                 2    13    0   16
COMPOOL SPACE USED =  1103

  0 ERRORS
```

```
$RUN OPT

FORT001010 UNABLE TO OPEN FILE
NAME    SEQ
MAIN.   00000
```

```
$RUN MACRO
MACRO V06-04A
*CCNTL,LP:<SYSYM,RPNML,MOSML,RPN,COD,OSEND
```

```
$JOB
DATE:-17-AUG-77
TIME:-00:46:32
$RUN PIP
PIP V10-03A
*WRITKW.SRC/DE

S257 000002
DK0: WRITKW.SRC

*DK0:WRITKW.SRC<BI:/FA

$EOD

*
$RUN MACRO
MACRO V06-04A
*WRITKW,LP:<SYSYM,WRITKW.SRC
```

TABLE OF CONTENTS

1- 2 SYSMPS MULTIPROCESSOR OPERATING SYSTEM

```
        .TITLE  SYSMPS  (MULTIPROCESSOR SYSTEM)
        .SBTTL  SYSMPS  MULTIPROCESSOR OPERATING SYSTEM
```

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 00:46 PAGE 4
SYSMPS  MULTIPROCESSOR OPERATING SYSTEM

                        ; SUBROUTINE TO WRITE KEYBOARD AND WAIT (TASK SW)
          000005                MSTK = %5
                                .GLOBL  WRTKW
  000000               WRTKW:
  000000 172425                 LDF     (MSTK)+,AC0        ; CONVERT TO FIXED
  000002 175445                 STCFI   AC0,-(MSTK)        ;
  000004 005725                 TST     (MSTK)+
  000006 012567                 MOV     (MSTK)+,BCK        ;PUT BYTE COUNT IN DEVB
         000162
  000012 172425                 LDF     (MSTK)+,AC0        ; CONVERT TO FIXED
  00014 175445                  STCFI   AC0,-(MSTK)        ;
  00016 005725                  TST     (MSTK)+
  00020 010046                  MOV     R0,-(SP)
  00022 010146                  MOV     R1,-(SP)
  00024 010246                  MOV     R2,-(SP)
  00026 012500                  MOV     (MSTK)+,R0         ;GET BUFFER ADDRESS
  00030 012702                  MOV     #LBUF,R2
         000200'
  00034 016701                  MOV     BCK,R1
         000134
  00040                1$:
  00040 112022                  MOVB    (R0)+,(R2)+        ;MOVE DATA TO LOCAL BUFFER
  00042 077102                  SOB     R1,1$
  00044 016701                  MOV     BCK,R1             ;MAX OF 80 CHAR. LINE
         000124
  00050 022701                  CMP     #79.,R1
         000117
  00054 003410                  BLE     2$
  00056 112722                  MOVB    #12,(R2)+          ;INSERT LF
         000012
  00062 112722                  MOVB    #15,(R2)+          ;INSERT CR
         000015
  00066 062767                  ADD     #2,BCK             ;CHANGE BYTE COUNT
         000002
         000100
  00074 000413                  BR      3$
  00076 162701 2$:              SUB     #78.,R1
         000116
  00102 005201                  INC     R1
  00104 160102                  SUB     R1,R2
  00106 112722                  MOVB    #12,(R2)+          ;INSERT LF
         000012
  00112 112722                  MOVB    #15,(R2)+          ;INSERT CR
         000015
  00116 012767                  MOV     #80.,BCK           ;CHANGE BYTE COUNT
         000120
         000050
  00124                3$:
  00124                         TESTIO  DEVBK,ERRK,3$      ;TEST I/O
  00134                         STRTIO  DEVBK,ERRK         ;WRITE TO DISK
  00142                         WAITIO  DEVBK,ERRK         ;WAIT (TASKSW)
  00150                ERRK:
  00150 012602                  MOV     (SP)+,R2           ; RESTORE REGISTERS
  00152 012601                  MOV     (SP)+,R1
  00154 012600                  MOV     (SP)+,R0
  00156 000207                  RTS     PC
  00160 000001 DEVBK:           .WORD   1                  ;FORWARD LINK
```

```
00162 000000          .WORD   0                ;PM #, PAGE
00164 000000          .WORD   0                ;DEV, UNIT
00166 000000          .WORD   0                ;STATUS
00170 000002          .WORD   2                ;FUNCTION
00172 000200'         .WORD   LBUF             ;@ BUFR
00174 000000  BCK:    .WORD   0                ;BYTE COUNT
00176 000000          .WORD   0                ;DDF
00200          LBUF:  .BLKW   40.              ;LOCAL BUF.
      000001          .END
```

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 00:46 PAGE 4-2
SYMBOL TABLE

AC0    =%000000          AC1    =%000001          AC2    =%000002
AC3    =%000003          AC4    =%000004          AC5    =%000005
BCK      000174R         B0     = 000001          B1     = 000002
B10    = 002000          B11    = 004000          B12    = 010000
B13    = 020000          B14    = 040000          B15    = 100000
B2     = 000004          B3     = 000010          B4     = 000020
B5     = 000040          B6     = 000100          B7     = 000200
B8     = 000400          B9     = 001000          CP     = 000005
DEVBK    000160R         DK     = 000001          D.BCNT= 000014
D.BUFR= 000012           D.DDF  = 000016          D.DEV  = 000004
D.FL   = 000000          D.FUNC= 000010           D.PAGE= 000003
D.PM   = 000002          D.STAT= 000006           D.UNIT= 000005
ERRK     000150R         EXIT   = 104411          I.ADDR= 000006
I.FL   = 000000          I.TIMR= 000002           I.TYPE= 000004
KB     = 000000          LBUF     000200R         LP     = 000004
MSTK   =%000005          MT     = 000002          M.DSA  = 000006
M.FLG  = 000003          M.FWA  = 000004          M.ICBT= 000014
M.ICBW= 000016           M.LGT  = 000010          M.NAM1= 000022
M.NAM2= 000024           M.OPT  = 000012          M.PH   = 000026
M.PROC= 000027           M.RUN  = 000001          M.SUSP= 000002
M.WAIT= 000000           M.WDT  = 000020          PAGE.0= 000000
PAGE.1= 000001           PAGE.2= 000002           PAGE.3= 000003
PAGE.4= 000004           PAGE.5= 000005           PAGE.6= 000006
PR     = 177000          PT     = 000006          P.AC0  = 000032
P.AC1  = 000042          P.AC2  = 000052          P.AC3  = 000062
P.AC4  = 000072          P.AC5  = 000102          P.BEG  = 000002
P.FLG  = 000026          P.FPS  = 000030          P.PC   = 000020
P.PGF  = 000024          P.PSW  = 000022          P.R0   = 000004
P.R1   = 000006          P.R2   = 000010          P.R3   = 000012
P.R4   = 000014          P.R5   = 000016          R6     =%000006
R7     =%000007          SWR    = 177570          TASKSW= 104410
TERM   = 104412          TM     = 000003          WAIT8  = 104404
WRTKW    000000RG        .TRAP  = 104400
. ABS.   000000      000
         000320      001
ERRORS DETECTED:  0
FREE CORE:  14073. WORDS
WRITKW,LP:<SYSYM,WRITKW.SRC


FREE CORE:  14073. WORDS
```

```
#
$FINISH
TIME:-00:47:34
```

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 00:48
TABLE OF CONTENTS

   1-  2  SYSMPS  MULTIPROCESSOR OPERATING SYSTEM
```

```
        .TITLE  SYSMPS  (MULTIPROCESSOR SYSTEM)
        .SBTTL  SYSMPS  MULTIPROCESSOR OPERATING SYSTEM
```

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 00:48 PAGE 4
SYSMPS  MULTIPROCESSOR OPERATING SYSTEM

1                      ; SUBROUTINE TO WRITE KEYBOARD AND NO WAIT (TASK SW)
2          000005              MSTK = %5
3                              .GLOBL  WRITK
4 000000              WRITK:
5 000000 172425                LDF     (MSTK)+,AC0       ; CONVERT TO FIXED
6 000002 175445                STCFI   AC0,-(MSTK)       ;
7 000004 005725                TST     (MSTK)+
8 000006 012567                MOV     (MSTK)+,BCK       ;PUT BYTE COUNT IN DEVB
         000154
9 000012 172425                LDF     (MSTK)+,AC0       ; CONVERT TO FIXED
10 00014 175445                STCFI   AC0,-(MSTK)       ;
11 00016 005725                TST     (MSTK)+
12 00020 010046                MOV     R0,-(SP)
13 00022 010146                MOV     R1,-(SP)
14 00024 010246                MOV     R2,-(SP)
15 00026 012500                MOV     (MSTK)+,R0        ;GET BUFFER ADDRESS
16 00030 012702                MOV     #LBUF,R2
         000172'
17 00034 016701                MOV     BCK,R1
         000126
18 00040              1$:
19 00040 112022                MOVB    (R0)+,(R2)+       ;MOVE DATA TO LOCAL BUFFER
20 00042 077102                SOB     R1,1$
21 00044 016701                MOV     BCK,R1            ;MAX OF 80 CHAR. LINE
         000116
22 00050 022701                CMP     #79.,R1
         000117
23 00054 003410                BLE     2$
24 00056 112722                MOVB    #12,(R2)+         ;INSERT LF
         000012
25 00062 112722                MOVB    #15,(R2)+         ;INSERT CR
         000015
26 00066 062767                ADD     #2,BCK            ;CHANGE BYTE COUNT
         000002
         000072
27 00074 000413                BR      3$
28 00076 162701  2$:           SUB     #78.,R1
         000116
29 00102 005201                INC     R1
30 00104 160102                SUB     R1,R2
31 00106 112722                MOVB    #12,(R2)+         ;INSERT LF
         000012
32 00112 112722                MOVB    #15,(R2)+         ;INSERT CR
         000015
33 00116 012767                MOV     #80.,BCK          ;CHANGE BYTE COUNT
         000120
         000042
34 00124              3$:
35 00124                       TESTIO  DEVBK,ERRK,3$     ;TEST I/O
36 00134                       STRTIO  DEVBK,ERRK        ;WRITE TO DISK
37 00142              ERRK:
38 00142 012602                MOV     (SP)+,R2          ; RESTORE REGISTERS
39 00144 012601                MOV     (SP)+,R1
40 00146 012600                MOV     (SP)+,R0
41 00150 000207                RTS     PC
42 00152 000001 DEVBK:        .WORD   1                 ;FORWARD LINK
43 00154 000000               .WORD   0                 ;PM #, PAGE
```

```
44 00156 000000               .WORD   0              ;DEV, UNIT
45 00160 000000               .WORD   0              ;STATUS
46 00162 000002               .WORD   2              ;FUNCTION
47 00164 000172'              .WORD   LBUF           ;@ BUFR
48 00166 000000 BCK:          .WORD   0              ;BYTE COUNT
49 00170 000000               .WORD   0              ;DDF
50 00172          LBUF:       .BLKW   40.            ;LOCAL BUF.
51        000001              .END
```

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 00:48 PAGE 4-2
SYMBOL TABLE

AC0     =%000000           AC1     =%000001           AC2     =%000002
AC3     =%000003           AC4     =%000004           AC5     =%000005
BCK      000166R           B0      = 000001           B1      = 000002
B10     = 002000           B11     = 004000           B12     = 010000
B13     = 020000           B14     = 040000           B15     = 100000
B2      = 000004           B3      = 000010           B4      = 000020
B5      = 000040           B6      = 000100           B7      = 000200
B8      = 000400           B9      = 001000           CP      = 000005
DEVBK    000152R           DK      = 000001           D.BCNT= 000014
D.BUFR= 000012             D.DDF  = 000016            D.DEV  = 000004
D.FL    = 000000           D.FUNC= 000010             D.PAGE= 000003
D.PM    = 000002           D.STAT= 000006             D.UNIT= 000005
ERBK     000142R           EXIT   = 104411            T.ADDR= 000006
I.FL    = 000000           I.TIMR= 000002             T.TYPE= 000004
KB      = 000000           LBUF     000172R           LP      = 000004
MSTK    =%000005           MT     = 000002            M.DSA  = 000006
M.FLG  = 000003            M.FWA  = 000004            M.TCBT= 000014
M.TCBW= 000016             M.LGT  = 000010            M.NAM1= 000022
M.NAM2= 000024             M.OPT  = 000012            M.PM   = 000026
M.PROC= 000027             M.RUN  = 000001            M.SUSP= 000002
M.WAIT= 000000             M.WDT  = 000020            PAGE.0= 000000
PAGE.1= 000001             PAGE.2= 000002             PAGE.3= 000003
PAGE.4= 000004             PAGE.5= 000005             PAGE.6= 000006
PR      = 177000           PT     = 000006            P.AC0  = 000032
P.AC1  = 000042            P.AC2  = 000052            P.AC3  = 000062
P.AC4  = 000072            P.AC5  = 000102            P.PEG  = 000002
P.FLG  = 000026            P.FPS  = 000030            P.PC   = 000020
P.PGE  = 000024            P.PSW  = 000022            P.R0   = 000004
P.R1   = 000006            P.R2   = 000010            P.R3   = 000012
P.R4   = 000014            P.R5   = 000016            R6     =%000006
R7      =%000007           SWR    = 177570            TASKSW= 104410
TERM   = 104412            TM     = 000003            WAITS  = 104404
WRITK    000000RG          .TRAP  = 104400
. ABS.   000000      000
         000312      001
ERRORS DETECTED:  0
FREE CORE:  13385. WORDS
WRITK,LP:<SYSYM,WRITK,SRC
```

SYSMPS (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 00:51
TABLE OF CONTENTS

1- 2 SYSMPS MULTIPROCESSOR OPERATING SYSTEM

```
        .TITLE  SYSMPS  (MULTIPROCESSOR SYSTEM)
        .SBTTL  SYSMPS  MULTIPROCESSOR OPERATING SYSTEM
```

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 00:51 PAGE 4
SYSMPS  MULTIPROCESSOR OPERATING SYSTEM

                       ; SUBROUTINE TO READ DISK, NO WAIT.
         000005                 MSTK = %5 .
                                .GLOBL  DISKR
  000000               DISKR:
  000000 172425                 LDF     (MSTK)+,AC0        ; CONVERT TO FIXED
  000002 175445                 STCFI   AC0,-(MSTK)        ;
  000004 005725                 TST     (MSTK)+
  000006 012567                 MOV     (MSTK)+,BUFRD
         000104
  000012 172425                 LDF     (MSTK)+,AC0        ; CONVERT TO FIXED
  00014 175445                  STCFI   AC0,-(MSTK)        ;
  00016 005725                  TST     (MSTK)+
  00020 012567                  MOV     (MSTK)+,SECT       ;ADDR OF SECTOR TO DEVB
        000076
  00024 172425                  LDF     (MSTK)+,AC0        ; CONVERT TO FIXED
  00026 175445                  STCFI   AC0,-(MSTK)        ;
  00030 005725                  TST     (MSTK)+
  00032 012567                  MOV     (MSTK)+,BCD        ;BYTE COUNT TO DEVB
        000062
  00036 005767                  TST     ICHK
        000004
  00042 001010                  BNE     IERR
  00044 000401                  BR      FRST
  00046 000000 ICHK:            .WORD   0
  00050        FRST:
  00050                         INITIO  DEVBD,IERR
  00056 012767                  MOV     #1,ICHK
        000001
        177762
  00064        IERR:
  00064        1$:              TESTIO  DEVBD,ERRD,1$
  00074                         STRTIO  DEVBD,ERRD
  00102        ERRD:
  00102 000207                  RTS     PC
  00104 000001 DEVBD:           .WORD   1                  ;FORWARD LINK
  00106 000000                  .WORD   0                  ;PM#, PAGE
  00110    001                  .BYTE   1,1                ;DEVICE, UNIT #
  00111    001
  00112 000000                  .WORD   0                  ;STATUS
  00114 000505                  .WORD   505                ;FUNCTION - READ
  00116 000000 BUFRD:           .WORD   0                  ;@BUFR
  00120 000000 BCD:             .WORD   0                  ;BYTE COUNT
  00122 000000 SECT:            .WORD   0                  ;SECTOR ADDR.
  00124 000000                  .WORD   0,0,0,0,0          ;RESERVED FOR DISK DRIVER.
  00126 000000
  00130 000000
  00132 000000
  00134 000000
        000001                  .END
```

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 00:51 PAGE 4-1
SYMBOL TABLE

AC0    =%000000          AC1    =%000001          AC2    =%000002
AC3    =%000003          AC4    =%000004          AC5    =%000005
BCD      000120R         BUFRD    000116R         B0     = 000001
B1     = 000002          B10    = 002000          B11    = 004000
B12    = 010000          B13    = 020000          B14    = 040000
B15    = 100000          B2     = 000004          B3     = 000010
B4     = 000020          B5     = 000040          B6     = 000100
B7     = 000200          B8     = 000400          B9     = 001000
CR     = 000005          DEVRD    000104R         DISKR    000000RG
DK     = 000001          D.BCNT= 000014           D.BUFR= 000012
D.DDF  = 000016          D.DEV  = 000004          D.FL   = 000000
D.FUNC= 000010           D.PAGE= 000003           D.PM   = 000002
D.STAT= 000006           D.UNIT= 000005           ERRD     000102R
EXIT   = 104111          ERST     000050R         ICHK     000046R
IERR     000064R         I.ADDR= 000006           I.FL   = 000000
I.TIMR= 000002           I.TYPE= 000004           KB     = 000000
LP     = 000004          MSTK   =%000005          MT     = 000002
M.DSA  = 000006          M.FLG  = 000003          M.FWA  = 000004
M.ICBT= 000014           M.ICBW= 000016           M.LGT  = 000010
M.NAM1= 000022           M.NAM2= 000024           M.OPT  = 000012
M.PM   = 000026          M.PROC= 000027           M.RUN  = 000001
M.SUSP= 000002           M.WAIT= 000000           M.WDT  = 000020
PAGE.0= 000000           PAGE.1= 000001           PAGE.2= 000002
PAGE.3= 000003           PAGE.4= 000004           PAGE.5= 000005
PAGE.6= 000006           PR     = 177000          PT     = 000006
P.AC0  = 000032          P.AC1  = 000042          P.AC2  = 000052
P.AC3  = 000062          P.AC4  = 000072          P.AC5  = 000102
P.BEG  = 000002          P.FLG  = 000026          P.FPS  = 000030
P.PC   = 000020          P.PGE  = 000024          P.PSW  = 000022
P.R0   = 000004          P.R1   = 000006          P.R2   = 000010
P.R3   = 000012          P.R4   = 000014          P.R5   = 000016
R6     =%000006          R7     =%000007          SECT     000122R
SWR    = 177570          TASKSW= 104410           TERM   = 104412
TM     = 000003          WAIT$  = 104404          .TRAP  = 104400

. ABS.  000000      000
        000136      001
ERRORS DETECTED:  0
FREE CORE:  13373. WORDS
DISKP,LP:<SYSYM,DISKR.SRC
```

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 00:53
TABLE OF CONTENTS

   1-  2  SYSMPS  MULTIPROCESSOR OPERATING SYSTEM
```

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 00:53 PAGE 1

                        .TITLE  SYSMPS  (MULTIPROCESSOR SYSTEM)
                        .SBTTL  SYSMPS  MULTIPROCESSOR OPERATING SYSTEM
```

```
                    ; SUBROUTINE TO WRITE TO PRINTER AND NO WAIT
        000005              MSTK = %5
                            .GLOBL  PRINT
000000              PRINT:
000000 172425               LDF     (MSTK)+,AC0         ; CONVERT TO FIXED
000002 175445               STCFI   AC0,-(MSTK)         ;
000004 005725               TST     (MSTK)+
000006 012567               MOV     (MSTK)+,BCPN        ;PUT BYTE COUNT IN DEVB
       000202
000012 172425               LDF     (MSTK)+,AC0         ; CONVERT TO FIXED
00014 175445                STCFI   AC0,-(MSTK)         ;
00016 005725                TST     (MSTK)+
00020 010046                MOV     R0,-(SP)
00022 010146                MOV     R1,-(SP)
00024 010246                MOV     R2,-(SP)
00026 012500                MOV     (MSTK)+,R0          ;GET BUFFER ADDRESS
00030 012702                MOV     #PBUF,P2
      000220'
00034 016701                MOV     BCPN,R1
      000154
00040               1$:
00040 112022                MOVB    (R0)+,(R2)+         ;MOVE DATA TO LOCAL BUFFER
00042 077102                SOB     R1,1$
00044 016701                MOV     BCPN,R1             ;MAX OF132 CHAR. LINE
      000144
00050 022701                CMP     #131.,R1
      000203
00054 003410                BLE     2$
00056 112722                MOVB    #12,(R2)+           ;INSERT LF
      000012
00062 112722                MOVB    #15,(R2)+           ;INSERT CR
      000015
00066 062767                ADD     #2,BCPN             ;CHANGE BYTE COUNT
      000002
      000120
00074 000413                BR      3$
00076 162701 2$:            SUB     #130.,R1
      000202
00102 005201                INC     R1
00104 160102                SUB     R1,R2
00106 112722                MOVB    #12,(R2)+           ;INSERT LF
      000012
00112 112722                MOVB    #15,(R2)+           ;INSERT CR
      000015
00116 012767                MOV     #132.,BCPN          ; CHANGE BYTE COUNT
      000204
      000070
00124               3$:
00124 005767                TST     ICHK
      000004
00130 001010                BNE     IERR
00132 000401                BR      FRST
00134 000000 ICHK:          .WORD   0
00136               FRST:
00136                       INITIO  DEVBP,IERR
00144 012767                MOV     #1,ICHK
      000001
```

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 00:53 PAGE 4-1
SYSMPS  MULTIPROCESSOR OPERATING SYSTEM

        177762
   00152          IERR:
   00152          3$:
   00152                   TESTIO  DEVBP,ERRP,3$     ;TEST I/O
   00162                   STRTIO  DEVBP,ERRP        ;WRITE TO PRINTER
   00170          ERRP:
   00170 012602            MOV     (SP)+,R2
   00172 012601            MOV     (SP)+,R1
   00174 012600            MOV     (SP)+,R0          ; RESTORE REGISTERS
   00176 000207            RTS     PC
   00200 000001   DEVBP:   .WORD   1                 ;FORWARD LINK
   00202 000000            .WORD   0                 ;PM #, PAGE
   00204    004            .BYTE   4,0               ; DEV,UNIT
   00205    000
   00206 000000            .WORD   0                 ;STATUS
   00210 000000            .WORD   0                 ;FUNCTION
   00212 000220'           .WORD   PBUF              ;@ BUFR
   00214 000000   BCPN:    .WORD   0                 ;BYTE COUNT
   00216 000000            .WORD   0                 ;DDF
   00220          PBUF:    .BLKW   66.               ;LOCAL BUF.
         000001            .END
```

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 00:53 PAGE 4-2
SYMBOL TABLE

AC0    =%000000        AC1    =%000001        AC2    =%000002
AC3    =%000003        AC4    =%000004        AC5    =%000005
BCPN     000214R       B0     = 000001        B1     = 000002
B10    = 002000        B11    = 004000        B12    = 010000
B13    = 020000        B14    = 040000        B15    = 100000
B2     = 000004        B3     = 000010        B4     = 000020
B5     = 000040        B6     = 000100        B7     = 000200
B8     = 000400        B9     = 001000        CP     = 000005
DEVBP    000200R       DK     = 000001        D.BCNT= 000014
D.BUFR= 000012         D.DDF = 000016         D.DEV = 000004
D.FL   = 000000        D.FUNC= 000010         D.PAGE= 000003
D.PM   = 000002        D.STAT= 000006         D.UNIT= 000005
ERRP     000170R       EXIT   = 104411        FRST     000136R
ICHK     000134R       IERR     000152R       I.ADDR= 000006
I.FL   = 000000        I.TIMR= 000002         I.TYPE= 000004
KB     = 000000        LP     = 000004        MSTK   =%000005
MT     = 000002        M.DSA = 000006         M.FLG = 000003
M.FWA = 000004         M.ICBT= 000014         M.ICBW= 000016
M.LGT = 000010         M.NAM1= 000022         M.NAM2= 000024
M.OPI = 000012         M.PM   = 000026        M.PROC= 000027
M.RUN = 000001         M.SUSP= 000002         M.WAIT= 000000
M.WDT = 000020         PAGE.0= 000000         PAGE.1= 000001
PAGE.2= 000002         PAGE.3= 000003         PAGE.4= 000004
PAGE.5= 000005         PAGE.6= 000006         PBUF     000220R
PR     = 177000        PRINT    000000RG      PT     = 000006
P.AC0 = 000032         P.AC1 = 000042         P.AC2 = 000052
P.AC3 = 000062         P.AC4 = 000072         P.AC5 = 000102
P.REG = 000002         P.FLG = 000026         P.FPS = 000030
P.PC   = 000020        P.PGE = 000024         P.PSW = 000022
P.R0   = 000004        P.R1   = 000006        P.R2   = 000010
P.R3   = 000012        P.R4   = 000014        P.R5   = 000016
R6     =%000006        R7     =%000007        SWR    = 177570
TASKSW= 104410         TERM   = 104412        TM     = 000003
WAITS = 104404         .TRAP = 104400
. ABS.   000000      000
         000424      001
ERRORS DETECTED:  0
FREE CORE:  13369. WORDS
PRINT,LP:<SYSYM,PRINT.SRC
```

SYSMPS (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 00:54
TABLE OF CONTENTS

1- 2 SYSMPS MULTIPROCESSOR OPERATING SYSTEM

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 00:54 PAGE 1

                    .TITLE  SYSMPS  (MULTIPROCESSOR SYSTEM)
                    .SBTTL  SYSMPS  MULTIPROCESSOR OPERATING SYSTEM
```

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 00:54 PAGE 4
SYSMPS  MULTIPROCESSOR OPERATING SYSTEM

                        ; SUBROUTINE TO READ A CARD AND NO WAIT
         000005                 MSTK = %5
                                .GLOBL  READC
  000000                READC:
  000000 172425                 LDF     (MSTK)+,AC0       ; CONVERT TO FIXED
  000002 175445                 STCFI   AC0,-(MSTK)       ;
  000004 005725                 TST     (MSTK)+
  000006 012567                 MOV     (MSTK)+,BUFRR     ;ADDR OF BUFFER TO DEVB
         000072
  000012 172425                 LDF     (MSTK)+,AC0       ; CONVERT TO FIXED
  00014 175445                  STCFI   AC0,-(MSTK)       ;
  00016 005725                  TST     (MSTK)+
  00020 012567                  MOV     (MSTK)+,BCR       ;BYTE COUNT TO DEVB
         000062
  00024 005767                  TST     ICHK
         000004
  00030 001010                  BNE     IERR
  00032 000401                  BR      FRST
  00034 000000  ICHK:           .WORD   0
  00036         FRST:
  00036                         INITIO  DEVBR,IERR
  00044 012767                  MOV     #1,ICHK
         000001
         177762
  00052         IERR:
  00052         1$:
  00052                         TESTIO  DEVBR,ERRR,1$
  00062                         STRTIO  DEVBR,ERRR
  00070         ERRR:
  00070 000207                  RTS     PC
  00072 000001  DEVBR:          .WORD   1                 ;FORWARD LINK
  00074 000000                  .WORD   0                 ;PN*, PAGE
  00076    005                  .BYTE   5,0               ;DEV, UNIT
  00077    000
  00100 000000                  .WORD   0                 ;STATUS
  00102 000000                  .WORD   0                 ;FUNCTION
  00104 000000  BUFRR:          .WORD   0                 ;@ BUFR
  00106 000000  BCR:            .WORD   0                 ;BYTE COUNT
  00110 000000                  .WORD   0                 ;DDF
         000001                 .END
```

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 00:54 PAGE 4-1
SYMBOL TABLE

AC0     =%000000          AC1     =%000001          AC2     =%000002
AC3     =%000003          AC4     =%000004          AC5     =%000005
BCR      000106R          BUFRR    000104R          B0      = 000001
B1      = 000002          B10     = 002000          B11     = 004000
B12     = 010000          B13     = 020000          B14     = 040000
B15     = 100000          B2      = 000004          B3      = 000010
B4      = 000020          B5      = 000040          B6      = 000100
B7      = 000200          B8      = 000400          B9      = 001000
CR      = 000005          DEVRR    000072R          DK      = 000001
D.BCNT= 000014            D.BUFR= 000012            D.DDF = 000016
D.DEV = 000004            D.FL  = 000000            D.FUNC= 000010
D.PAGE= 000003            D.PM  = 000002            D.STAT= 000006
D.UNIT= 000005            ERRR     000070R          EXIT  = 104411
FRST     000035R          ICHK     000034R          IERR     000052R
I.ADDR= 000006            I.FL  = 000000            I.TIMP= 000002
I.TYPE= 000004            KB    = 000000            LP    = 000004
MSTK  =%000005            MT    = 000002            M.DSA = 000006
M.FLG = 000003            M.FWA = 000004            M.ICBT= 000014
M.ICBW= 000016            M.LGT = 000010            M.NAM1= 000022
M.NAM2= 000024            M.OPT = 000012            M.PM  = 000026
M.PROC= 000027            M.RUN = 000001            M.SUSP= 000002
M.WAIT= 000000            M.WDT = 000020            PAGE.0= 000000
PAGE.1= 000001            PAGE.2= 000002            PAGE.3= 000003
PAGE.4= 000004            PAGE.5= 000005            PAGE.6= 000006
PR    = 177000            PT    = 000006            P.AC0 = 000032
P.AC1 = 000042            P.AC2 = 000052            P.AC3 = 000062
P.AC4 = 000072            P.AC5 = 000102            P.BEG = 000002
P.FLG = 000026            P.FPS = 000030            P.PC  = 000020
P.PGE = 000024            P.PSW = 000022            P.R0  = 000004
P.R1  = 000006            P.R2  = 000010            P.R3  = 000012
P.R4  = 000014            P.R5  = 000016            READC    000000RG
R6    =%000006            R7    =%000007            SWR   = 177570
TASKSW= 104410            TERM  = 104412            TM    = 000003
WAITS = 104404            .TRAP = 104400
. ABS.   000000    000
         000112    001
ERRORS DETECTED:  0
FREE CORE:  13385. WORDS
READC,LP:<SYSYM,READC.SRC
```

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 00:55
TABLE OF CONTENTS

   1-  2  SYSMPS  MULTIPROCESSOR OPERATING SYSTEM
```

```
        .TITLE  SYSMPS  (MULTIPROCESSOR SYSTEM)
        .SBTTL  SYSMPS  MULTIPROCESSOR OPERATING SYSTEM
```

```
                          ; SUBROUTINE TO WRITE TO DISK, NO WAIT.
          000005                  MSTK = %5
                                  .GLOBL  DISKW
  000000               DISKW:
  000000  172425                  LDF     (MSTK)+,AC0         ; CONVERT TO FIXED
  000002  175445                  STCFI   AC0,-(MSTK)         ;
  000004  005725                  TST     (MSTK)+
  000006  012567                  MOV     (MSTK)+,BUFRD
          000104
  000012  172425                  LDF     (MSTK)+,AC0         ; CONVERT TO FIXED
  00014   175445                  STCFI   AC0,-(MSTK)         ;
  00016   005725                  TST     (MSTK)+
  00020   012567                  MOV     (MSTK)+,SECT        ;ADDR OF SECTOR TO DEVB
          000076
  00024   172425                  LDF     (MSTK)+,AC0         ; CONVERT TO FIXED
  00026   175445                  STCFI   AC0,-(MSTK)         ;
  00030   005725                  TST     (MSTK)+
  00032   012567                  MOV     (MSTK)+,BCD         ;BYTE COUNT TO DEVB
          000062
  00036   005767                  TST     ICHK
          000004
  00042   001010                  BNE     IERR
  00044   000401                  BR      FRST
  00046   000000       ICHK:      .WORD   0
  00050                FRST:
  00050                           INITIO  DEVBD,IERR
  00056   012767                  MOV     #1,ICHK
          000001
          177762
  00064                IERR:
  00064                1$:        TESTIO  DEVBD,ERRD,1$
  00074                           STRTIO  DEVBD,ERRD
  00102                ERRD:
  00102   000207                  RTS     PC
  00104   000001       DEVBD:     .WORD   1                   ;FORWARD LINK
  00106   000000                  .WORD   0                   ;PM#, PAGE
  00110      001                  .BYTE   1,1                 ;DEVICE, UNIT #
  00111      001
  00112   000000                  .WORD   0                   ;STATUS
  00114   000503                  .WORD   503                 ;FUNCTION - WRITE
  00116   000000       BUFRD:     .WORD   0                   ;@BUFR
  00120   000000       BCD:       .WORD   0                   ;BYTE COUNT
  00122   000000       SECT:      .WORD   0                   ;SECTOR ADDR.
  00124   000000                  .WORD   0,0,0,0,0           ;RESERVED FOR DISK DRIVER.
  00126   000000
  00130   000000
  00132   000000
  00134   000000
          000001                  .END
```

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 00:55 PAGE 4-1
SYMBOL TABLE

AC0     =%000000          AC1     =%000001          AC2     =%000002
AC3     =%000003          AC4     =%000004          AC5     =%000005
BCD      000120R          BUFRD    000116R          B0      = 000001
B1      = 000002          B10     = 002000          B11     = 004000
B12     = 010000          B13     = 020000          B14     = 040000
B15     = 100000          B2      = 000004          B3      = 000010
B4      = 000020          B5      = 000040          B6      = 000100
B7      = 000200          B8      = 000400          B9      = 001000
CR      = 000005          DEVRD    000104R          DISKW    000000RG
DK      = 000001          D.BCNT= 000014            D.BUFR= 000012
D.DDF = 000016            D.DEV = 000004            D.FL    = 000000
D.FUNC= 000010            D.PAGE= 000003            D.PM    = 000002
D.STAT= 000006            D.UNIT= 000005            ERRD     000102R
EXIT    = 104411          FRST     000050R          ICHK     000046R
IERR     000064R          I.ADDR= 000006            I.FL    = 000000
I.TIMR= 000002            I.TYPE= 000004            KB      = 000000
LP      = 000004          MSTK    =%000005          MT      = 000002
M.DSA = 000006            M.FLG = 000003            M.FWA = 000004
M.ICBT= 000014            M.ICBW= 000016            M.IGT = 000010
M.NAM1= 000022            M.NAM2= 000024            M.OPT = 000012
M.PM    = 000026          M.PROC= 000027            M.RUN = 000001
M.SUSP= 000002            M.WAIT= 000000            M.WDT = 000020
PAGE.0= 000000            PAGE.1= 000001            PAGE.2= 000002
PAGE.3= 000003            PAGE.4= 000004            PAGE.5= 000005
PAGE.6= 000006            PR      = 177000          PT      = 000006
P.AC0 = 000032            P.AC1 = 000042            P.AC2 = 000052
P.AC3 = 000062            P.AC4 = 000072            P.AC5 = 000102
P.BEG = 000002            P.FLG = 000026            P.FPS = 000030
P.PC    = 000020          P.PGE = 000024            P.PSW = 000022
P.R0    = 000004          P.R1    = 000006          P.R2    = 000010
P.R3    = 000012          P.R4    = 000014          P.R5    = 000016
R6      =%000006          R7      =%000007          SECT     000122R
SWR     = 177570          TASKSW= 104410            TERM    = 104412
TM      = 000003          WAITS = 104404            .TRAP = 104400

. ABS.  000000    000
        000136    001
ERRORS DETECTED:  0
FREE CORE:  13373. WORDS
DISKW,LP:<SYSYM,DISKW.SRC
```

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 00:56
TABLE OF CONTENTS

   1-   2  SYSMPS  MULTIPROCESSOR OPERATING SYSTEM
```

```
        .TITLE  SYSMPS  (MULTIPROCESSOR SYSTEM)
        .SBTTL  SYSMPS  MULTIPROCESSOR OPERATING SYSTEM
```

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 00:56
TABLE OF CONTENTS

   1-  2  SYSMPS  MULTIPROCESSOR OPERATING SYSTEM
```

```
                .TITLE  SYSMPS  (MULTIPROCESSOR SYSTEM)
                .SBTTL  SYSMPS  MULTIPROCESSOR OPERATING SYSTEM
```

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 00:58 PAGE 4
SYSMPS  MULTIPROCESSOR OPERATING SYSTEM

1                     ; SUBROUTINE TO WRITE TO PRINTER AND WAIT
2           000005            MSTK = %5
3                             .GLOBL  PRNTW
4 000000              PRNTW:
5 000000 172425               LDF     (MSTK)+,AC0       ; CONVERT TO FIXED
6 000002 175445               STCFI   AC0,-(MSTK)       ;
7 000004 005725               TST     (MSTK)+
8 000006 012567               MOV     (MSTK)+,BCP       ;PUT BYTE COUNT IN DEVR
         000210
9 000012 172425               LDF     (MSTK)+,AC0       ; CONVERT TO FIXED
10 00014 175445               STCFI   AC0,-(MSTK)       ;
11 00016 005725               TST     (MSTK)+
12 00020 010046               MOV     R0,-(SP)
13 00022 010146               MOV     R1,-(SP)
14 00024 010246               MOV     R2,-(SP)
15 00026 012500               MOV     (MSTK)+,R0        ;GET BUFFER ADDRESS
16 00030 012702               MOV     #PBUF,R2
         000226'
17 00034 016701               MOV     BCP,R1
         000162
18 00040              1$:
19 00040 112022               MOVB    (R0)+,(R2)+       ;MOVE DATA TO LOCAL BUFFER
20 00042 077102               SOB     R1,1$
21 00044 016701               MOV     BCP,R1            ;MAX OF 80 CHAR. LINE
         000152
22 00050 022701               CMP     #131.,R1
         000203
23 00054 003410               BLE     2$
24 00056 112722               MOVB    #12,(R2)+         ;INSERT LF
         000012
25 00062 112722               MOVB    #15,(R2)+         ;INSERT CR
         000015
26 00066 062767               ADD     #2,BCP            ;CHANGE BYTE COUNT
         000002
         000126
27 00074 000413               BR      3$
28 00076 162701 2$:           SUB     #130.,R1
         000202
29 00102 005201               INC     R1
30 00104 160102               SUB     R1,R2
31 00106 112722               MOVB    #12,(R2)+         ;INSERT LF
         000012
32 00112 112722               MOVB    #15,(R2)+         ;INSERT CR
         000015
33 00116 012767               MOV     #132.,BCP         ;CHANGE BYTE COUNT
         000204
         000076
34 00124              3$:
35 00124 005767               TST     ICHK
         000004
36 00130 001010               BNE     IERR
37 00132 000401               BR      FRST
38 00134 000000 ICHK:         .WORD   0
39 00136        FRST:
40 00136                      INITIO  DEVRP,IERR
41 00144 012767               MOV     #1,ICHK
         000001
```

```
         177762
42 00152            TERR:
43 00152            3$:
44 00152                    TESTIO  DEVBP,ERRP,3$     ;TEST I/O
45 00162                    STRTIO  DEVBP,ERRP        ;WRITE TO PRINTER
46 00170                    WAITIO  DEVBP,ERRP        ;WAIT (TASKSW)
47 00176            ERRP:
48 00176 012602             MOV     (SP)+,R2          ; RESTORE REGISTERS
49 00200 012601             MOV     (SP)+,R1
50 00202 012600             MOV     (SP)+,R0
51 00204 000207             RTS     PC
52 00206 000001     DEVBP:  .WORD   1                 ;FORWARD LINK
53 00210 000000             .WORD   0                 ;PM #, PAGE
54 00212    004             .BYTE   4,0               ; DEV,UNIT
   00213    000
55 00214 000000             .WORD   0                 ; STATUS
56 00216 000000             .WORD   0                 ;FUNCTION
57 00220 000226'            .WORD   PBUF              ;@ BUFR
58 00222 000000     BCP:    .WORD   0                 ;BYTE COUNT
59 00224 000000             .WORD   0                 ;DDF
60 00226            PBUF:   .BLKW   66.               ;LOCAL BUF.
61       000001             .END
```

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 00:58 PAGE 4-2
SYMBOL TABLE

AC0     =%000000          AC1     =%000001          AC2     =%000002
AC3     =%000003          AC4     =%000004          AC5     =%000005
BCP      000222R          B0      = 000001          B1      = 000002
B10     = 002000          B11     = 004000          B12     = 010000
B13     = 020000          B14     = 040000          B15     = 100000
B2      = 000004          B3      = 000010          B4      = 000020
B5      = 000040          B6      = 000100          B7      = 000200
B8      = 000400          B9      = 001000          CR      = 000005
DEVBP    000206R          DK      = 000001          D.BCNT= 000014
D.BUFR= 000012            D.DDF = 000016            D.DEV = 000004
D.FL  = 000000            D.FUNC= 000010            D.PAGE= 000003
D.PM  = 000002            D.STAT= 000006            D.UNIT= 000005
ERRP     000176R          EXIT  = 104411            FRST     000136R
ICHK     000134R          IERR     000152R          I.ADDR= 000006
I.FL  = 000000            I.TIMR= 000002            I.TYPE= 000004
KB    = 000000            LP    = 000004            MSTK  =%000005
MT    = 000002            M.DSA = 000006            M.FLG = 000003
M.FWA = 000004            M.ICBT= 000014            M.ICBK= 000016
M.LGT = 000010            M.NAM1= 000022            M.NAM2= 000024
M.OPT = 000012            M.PM  = 000026            M.PROC= 000027
M.RUN = 000001            M.SUSP= 000002            M.WAIT= 000000
M.WDT = 000020            PAGE.0= 000000            PAGE.1= 000001
PAGE.2= 000002            PAGE.3= 000003            PAGE.4= 000004
PAGE.5= 000005            PAGE.6= 000006            PBUF     000226R
PP    = 177000            PRNTW    000000G          PT    = 000006
P.AC0 = 000032            P.AC1 = 000042            P.AC2 = 000052
P.AC3 = 000062            P.AC4 = 000072            P.AC5 = 000102
P.BEG = 000002            P.FLG = 000026            P.FPS = 000030
P.PC  = 000020            P.PGE = 000024            P.PSW = 000022
P.R0  = 000004            P.R1  = 000006            P.R2  = 000010
P.R3  = 000012            P.R4  = 000014            P.R5  = 000016
R6    =%000006            R7    =%000007            SWR   = 177570
TASKSW= 104410            TERM  = 104412            TM    = 000003
WAITS = 104404            .TRAP = 104400
. ABS.   000000     000
         000432     001
ERRORS DETECTED:  0
FREE CORE:  13369. WORDS
PRINTW,LP:<SYSYM,PRINTW.SRC
```

TABLE OF CONTENTS

1- 2 SYSMPS MULTIPROCESSOR OPERATING SYSTEM

```
        .TITLE  SYSMPS  (MULTIPROCESSOR SYSTEM)
        .SBTTL  SYSMPS  MULTIPROCESSOR OPERATING SYSTEM
```

```
1                      ; SUBROUTINE TO READ DISK & WAIT.
2          000005                 MSTK = %5
3                                 .GLOBL  DSKRW
4 000000           DSKRW:
5 000000 172425                   LDF     (MSTK)+,AC0        ; CONVERT TO FIXED
6 000002 175445                   STCFI   AC0,-(MSTK)        ;
7 000004 005725                   TST     (MSTK)+
8 000006 012567                   MOV     (MSTK)+,BUFRD      ;ADDR OF OUTPUT BUFR TO DEVB
         000112
9 000012 172425                   LDF     (MSTK)+,AC0        ; CONVERT TO FIXED
10 00014 175445                   STCFI   AC0,-(MSTK)        ;
11 00016 005725                   TST     (MSTK)+
12 00020 012567                   MOV     (MSTK)+,SECT       ;ADDR OF SECTOR TO DEVB
         000104
13 00024 172425                   LDF     (MSTK)+,AC0        ; CONVERT TO FIXED
14 00026 175445                   STCFI   AC0,-(MSTK)        ;
15 00030 005725                   TST     (MSTK)+
16 00032 012567                   MOV     (MSTK)+,BCD        ;BYTE COUNT TO DEVB
         000070
17 00036 005767                   TST     ICHK
         000004
18 00042 001010                   BNE     IERR
19 00044 000401                   BR      FRST
20 00046 000000 ICHK:             .WORD   0
21 00050        FRST:
22 00050                          INITIO  DEVBD,IERR
23 00056 012767                   MOV     #1,ICHK
         000001
         177762
24 00064        IERR:
25 00064        1$:               TESTIO  DEVBD,ERRD,1$
26 00074                          STRTIO  DEVBD,ERRD         ;WRITE DISK TO BUFFER
27 00102                          WAITIO  DEVBD,ERRD
28 00110        ERRD:
29 00110 000207                   RTS     PC
30 00112 000001 DEVBD:            .WORD   1                  ;FORWARD LINK
31 00114 000000                   .WORD   0                  ;PN#, PAGE
32 00116    001                   .BYTE   1,1                ;DEVICE, UNIT #
   00117    001
33 00120 000000                   .WORD   0                  ;STATUS
34 00122 000505                   .WORD   505                ;FUNCTION - READ
35 00124 000000 BUFRD:            .WORD   0                  ;@BUFR
36 00126 000000 BCD:              .WORD   0                  ;BYTE COUNT
37 00130 000000 SECT:             .WORD   0                  ;SECTOR ADDR.
38 00132 000000                   .WORD   0,0,0,0,0          ;RESERVED FOR DISK DRIVER.
   00134 000000
   00136 000000
   00140 000000
   00142 000000
39       000001                   .END
```

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 01:06 PAGE 1

                        .TITLE  SYSMPS  (MULTIPROCESSOR SYSTEM)
                        .SBTTL  SYSMPS  MULTIPROCESSOR OPERATING SYSTEM
```

```
                      ;-------------------------------------------------------------------
                      ;      THIS PROGRAM READS FROM THE LA36 KEYBOARD AND STORES THE INPUT DAT
                      ;      IN THE DESIGNATED BUFFER.  THE CALL IS :
                      ;      PUSH      ADDRS OF INPUT BUFFER
                      ;      PUSH      ADDR OF ACTUAL BYTE COUNT (RETURNED)
                      ;      JSR       R4,REDKW
                      ;-------------------------------------------------------------------
         000005               MSTK = %5
                              .GLOBL  READK
00000                 READK:
00000    172425               LDF     (MSTK)+,AC0       ; CONVERT TO FIXED
00002    175445               STCFI   AC0,-(MSTK)       ;
00004    005725               TST     (MSTK)+           ;
00006    012567               MOV     (MSTK)+,BUFRK     ; ADDRESS OF INPUT BUFFER TO DEVB
         000052
00012    172425               LDF     (MSTK)+,AC0       ; CONVERT TO FIXED
00014    175445               STCFI   AC0,-(MSTK)       ;
00016    005725               TST     (MSTK)+           ;
00020    010446               MOV     R4,-(SP)          ; SAVE R4
00022    012504               MOV     (MSTK)+,R4        ; PUT ADDR TO RETURN ACTUAL BC INTO R4
00024                 1$:     TESTIO  DEVBRK,KERR,1$    ; TEST I/O DRIVER FOR KB
00034                         STRTIO  DEVBRK,KERR       ; BEGIN READ
00042                 KERR:
00042    016714               MOV     DEVBRK+16,@R4     ; PUT ACTUAL BC INTO DESIGNATED ADDR
         000022
00046    012604               MOV     (SP)+,R4          ; RESTORE R4
00050    000207               RTS     PC
00052    000001       DEVBRK: .WORD   1                 ; FORWARD LINK
00054       000               .BYTE   0,0               ; PM#,PAGE #
00055       000
00056       000               .BYTE   0,0               ; DEV,UNIT #
00057       000
00060    000000               .WORD   0                 ; STATUS
00062    000001               .WORD   1                 ; FUNCTION
00064    000000       BUFRK:  .WORD   0                 ; @ BUFR
00066    000120               .WORD   80.               ; BYTE COUNT
00070    000000               .WORD   0                 ; DDF
         000001               .END
```

AD-A054 508 HARRIS CORP MELBOURNE FLA ELECTRONIC SYSTEMS DIV F/G 9/2

INTELLIGENCE SECURITY SUBSYSTEM.(U)

MAR 78 F ANDERS, W MALL, R MCGILL F30602-76-C-0445

UNCLASSIFIED RADC-TR-78-33 NL

4 OF 4

AD A054508

END

DATE FILMED

6-78

DDC

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 01:06 PAGE 4-1
SYMBOL TABLE

AC0    =%000000          AC1    =%000001          AC2    =%000002
AC3    =%000003          AC4    =%000004          AC5    =%000005
BUFBK   000064R          B0     = 000001          B1     = 000002
B10    = 002000          B11    = 004000          B12    = 010000
B13    = 020000          B14    = 040000          B15    = 100000
B2     = 000004          B3     = 000010          B4     = 000020
B5     = 000040          B6     = 000100          B7     = 000200
B8     = 000400          B9     = 001000          CR     = 000005
DEVBRK  000052R          DK     = 000001          D.BCNT= 000014
D.BUFR= 000012           D.DDF  = 000016          D.DEV  = 000004
D.FL   = 000000          D.FUNC= 000010           D.PAGE= 000003
D.PM   = 000002          D.STAT= 000006           D.UNIT= 000005
EXIT   = 104411          I.ADDR= 000006           I.FL   = 000000
I.TIMR= 000002           I.TYPE= 000004           KB     = 000000
KERR    000042R          LP     = 000004          MSTK   =%000005
MT     = 000002          M.DSA  = 000006          M.FLG  = 000003
M.FWA  = 000004          M.TCBT= 000014           M.ICBW= 000016
M.LGT  = 000010          M.NAM1= 000022           M.NAM2= 000024
M.OPI  = 000012          M.PM   = 000026          M.PROC= 000027
M.RUN  = 000001          M.SUSP= 000002           M.WAIT= 000000
M.WDT  = 000020          PAGE.0= 000000           PAGE.1= 000001
PAGE.2= 000002           PAGE.3= 000003           PAGE.4= 000004
PAGE.5= 000005           PAGE.6= 000006           PP     = 177000
PT     = 000006          P.AC0  = 000032          P.AC1  = 000042
P.AC2  = 000052          P.AC3  = 000062          P.AC4  = 000072
P.AC5  = 000102          P.BEG  = 000002          P.FLG  = 000026
P.FPS  = 000030          P.PC   = 000020          P.PGF  = 000024
P.PSW  = 000022          P.R0   = 000004          P.R1   = 000006
P.R2   = 000010          P.R3   = 000012          P.R4   = 000014
P.R5   = 000016          READK   000000RG         R6     =%000006
R7     =%000007          SWR    = 177570          TASKSW= 104410
TERM   = 104412          TM     = 000003          WAIT8  = 104404
.TRAP  = 104400
. ABS.  000000      000
        000072      001
ERRORS DETECTED:  0
FREE CORE:  13401. WORDS
READK,LP:<SYSYM,READK.SR
```

TABLE OF CONTENTS

1- 2 SYSMPS MULTIPROCESSOR OPERATING SYSTEM

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 01:07 PAGE 1

                        .TITLE  SYSMPS  (MULTIPROCESSOR SYSTEM)
                        .SBTTL  SYSMPS  MULTIPROCESSOR OPERATING SYSTEM
```

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 01:07 PAGE 4
SYSMPS  MULTIPROCESSOR OPERATING SYSTEM

                    ;-----------------------------------------------------------------------
                    ;       THIS PROGRAM READS FROM THE LA36 KEYBOARD AND STORES THE INPUT DAT
                    ;       IN THE DESIGNATED BUFFER.  THE CALL IS :
                    ;       PUSH      ADDRS OF INPUT BUFFER
                    ;       PUSH      ADDR OF ACTUAL BYTE COUNT (RETURNED)
                    ;       JSR       R4,REDKW
                    ;-----------------------------------------------------------------------
           000005           MSTK = %5
                            .GLOBL  REDKW
   00000            REDKW:
   00000 172425             LDF     (MSTK)+,AC0       ; CONVERT TO FIXED
   00002 175445             STCFI   AC0,-(MSTK)       ;
   00004 005725             TST     (MSTK)+           ;
   00006 012567             MOV     (MSTK)+,BUFRK     ; ADDRESS OF INPUT BUFFER TO DEVB
         000060
   00012 172425             LDF     (MSTK)+,AC0       ; CONVERT TO FIXED
   00014 175445             STCFI   AC0,-(MSTK)       ;
   00016 005725             TST     (MSTK)+           ;
   00020 010446             MOV     R4,-(SP)          ; SAVE R4
   00022 012504             MOV     (MSTK)+,R4        ; PUT ADDR TO RETURN ACTUAL BC INTO R4
   00024            1$:     TESTIO  DEVBRK,KERR,1$    ; TEST I/O DRIVER FOR KB
   00034                    STRTIO  DEVBRK,KERR       ; BEGIN READ
   00042                    WAITIO  DEVBRK,KERR       ; WAIT FOR I/O TO COMPLETE
   00050            KERR:
   00050 016714             MOV     DEVBRK+16,@R4     ; PUT ACTUAL BC INTO DESIGNATED ADDR
         000022
   00054 012604             MOV     (SP)+,R4          ; RESTORE R4
   00056 000207             RTS     PC
   00060 000001 DEVBRK:     .WORD   1                 ; FORWARD LINK
   00062    000             .BYTE   0,0               ; PK#,PAGE #
   00063    000
   00064    000             .BYTE   0,0               ; DEV,UNIT #
   00065    000
   00066 000000             .WORD   0                 ; STATUS
   00070 000001             .WORD   1                 ; FUNCTION
   00072 000000 BUFRK:      .WORD   0                 ; @ BUFR
   00074 000120             .WORD   80.               ; BYTE COUNT
   00076 000000             .WORD   0                 ; DDF
           000001           .END
```

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 01:07 PAGE 4-1
SYMBOL TABLE

AC0    =%000000          AC1    =%000001          AC2    =%000002
AC3    =%000003          AC4    =%000004          AC5    =%000005
BUFBK    000072R         B0     = 000001          B1     = 000002
B10    = 002000          B11    = 004000          B12    = 010000
B13    = 020000          B14    = 040000          B15    = 100000
B2     = 000004          B3     = 000010          B4     = 000020
B5     = 000040          B6     = 000100          B7     = 000200
B8     = 000400          B9     = 001000          CP     = 000005
DEVBRK   000060R         DK     = 000001          D.BCNT= 000014
D.BUFR= 000012           D.DDF = 000016           D.DEV = 000004
D.FL   = 000000          D.FUNC= 000010           D.PAGE= 000003
D.PM   = 000002          D.STAT= 000006           D.UNIT= 000005
EXIT   = 104411          I.ADDR= 000006           I.FL   = 000000
I.TIMR= 000002           I.TYPE= 000004           KB     = 000000
KERR     000050R         LP     = 000004          MSTK   =%000005
MT     = 000002          M.DSA = 000006           M.FLG = 000003
M.FWA = 000004           M.ICBT= 000014           M.ICBW= 000016
M.LGT = 000010           M.NAM1= 000022           M.NAM2= 000024
M.OPT = 000012           M.PM  = 000026           M.PROC= 000027
M.PUH = 000001           M.SUSP= 000002           M.WAIT= 000000
M.WDT = 000020           PAGE.0= 000000           PAGE.1= 000001
PAGE.2= 000002           PAGE.3= 000003           PAGE.4= 000004
PAGE.5= 000005           PAGE.6= 000006           PR     = 177000
PT     = 000006          P.AC0 = 000032           P.AC1 = 000042
P.AC2 = 000052           P.AC3 = 000062           P.AC4 = 000072
P.AC5 = 000102           P.BEG = 000002           P.FLG = 000026
P.FPS = 000030           P.PC  = 000020           P.PGF = 000024
P.PSW = 000022           P.R0  = 000004           P.R1  = 000006
P.R2  = 000010           P.R3  = 000012           P.R4  = 000014
P.R5  = 000016           REDKW    000000RG        R6     =%000006
R7     =%000007          SWR    = 177570          TASKSW= 104410
TERM   = 104412          TM     = 000003          WAITS = 104404
.TRAP = 104400
. ABS.   000000      000
         000100      001
ERRORS DETECTED:  0
FREE CORE:  13401. WORDS
REDKW,LP:<SYSYM,REDKW.SRC
```

TABLE OF CONTENTS

1- 2 SYSMPS MULTIPROCESSOR OPERATING SYSTEM

```
                .TITLE  SYSMPS  (MULTIPROCESSOR SYSTEM)
                .SBTTL  SYSMPS  MULTIPROCESSOR OPERATING SYSTEM
```

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 01:09 PAGE 4
SYSMPS  MULTIPROCESSOR OPERATING SYSTEM

                     ; SUBROUTINE TO READ A CARD AND WAIT
        000005                MSTK = %5
                              .GLOBL  REDCW
 000000              REDCW:
 000000 172425                LDF     (MSTK)+,AC0       ; CONVERT TO FIXED
 000002 175445                STCFI   AC0,-(MSTK)       ;
 000004 005725                TST     (MSTK)+
 000006 012567                MOV     (MSTK)+,BUFRR     ;ADDR OF BUFFER TO DEVB
        000100
 000012 172425                LDF     (MSTK)+,AC0       ; CONVERT TO FIXED
 00014 175445                 STCFI   AC0,-(MSTK)       ;
 00016 005725                 TST     (MSTK)+
 00020 012567                 MOV     (MSTK)+,BCR       ;BYTE COUNT TO DEVB
        000070
 00024 005767                 TST     ICHK
        000004
 00030 001010                 BNE     IERR
 00032 000401                 BR      FRST
 00034 000000 ICHK:           .WORD   0
 00036              FRST:
 00036                        INITIO  DEVBR,IERR
 00044 012767                 MOV     #1,ICHK
        000001
        177762
 00052              IERR:
 00052              1$:
 00052                        TESTIO  DEVBR,ERRR,1$
 00062                        STRTIO  DEVBR,ERRR
 00070                        WAITIO  DEVBR,ERRR
 00076              ERRR:
 00076 000207                 RTS     PC
 00100 000001 DEVBR:          .WORD   1                 ;FORWARD LINK
 00102 000000                 .WORD   0                 ;PM#, PAGE
 00104    005                 .BYTE   5,0               ;DEV, UNIT
 00105    000
 00106 000000                 .WORD   0                 ;STATUS
 00110 000000                 .WORD   0                 ;FUNCTION
 00112 000000 BUFRR:          .WORD   0                 ;@ BUFR
 00114 000000 BCR:            .WORD   0                 ;BYTE COUNT
 00116 000000                 .WORD   0                 ;DDF
        000001                .END
```

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 17-AUG-77 01:09 PAGE 4-1
SYMBOL TABLE

AC0    =%000000          AC1    =%000001          AC2    =%000002
AC3    =%000003          AC4    =%000004          AC5    =%000005
BCR     000114R          BUFRR   000112R          B0     = 000001
B1     = 000002          B10    = 002000          B11    = 004000
B12    = 010000          B13    = 020000          B14    = 040000
B15    = 100000          B2     = 000004          B3     = 000010
B4     = 000020          B5     = 000040          B6     = 000100
B7     = 000200          B8     = 000400          B9     = 001000
CR     = 000005          DEVRR   000100R          DK     = 000001
D.BCNT= 000014           D.BUFR= 000012           D.DDF = 000016
D.DEV = 000004           D.FL  = 000000           D.FUNC= 000010
D.PAGE= 000003           D.PM  = 000002           D.STAT= 000006
D.UNIT= 000005           ERRR    000076R          EXIT  = 104411
FRST    000036R          ICHK    000034R          IERR    000052R
I.ADDR= 000006           I.FL  = 000000           I.TIMR= 000002
I.TYPE= 000004           KB    = 000000           LP    = 000004
MSTK  =%000005           MT    = 000002           M.DSA = 000006
M.FLG = 000003           M.FWA = 000004           M.ICRT= 000014
M.ICRW= 000016           M.LGT = 000010           M.NAM1= 000022
M.NAM2= 000024           M.OPT = 000012           M.PM  = 000026
M.PROC= 000027           M.RUN = 000001           M.SUSP= 000002
M.WAIT= 000000           M.VDT = 000020           PAGE.0= 000000
PAGE.1= 000001           PAGE.2= 000002           PAGE.3= 000003
PAGE.4= 000004           PAGE.5= 000005           PAGE.6= 000006
PP    = 177000           PT    = 000006           P.AC0 = 000032
P.AC1 = 000042           P.AC2 = 000052           P.AC3 = 000062
P.AC4 = 000072           P.AC5 = 000102           P.BFG = 000002
P.FLG = 000026           P.FPS = 000030           P.PC  = 000020
P.PGE = 000024           P.PSW = 000022           P.R0  = 000004
P.R1  = 000006           P.R2  = 000010           P.R3  = 000012
P.R4  = 000014           P.R5  = 000016           REDCW   000000RG
R6    =%000006           R7    =%000007           SWR   = 177570
TASKSW= 104410           TERM  = 104412           TM    = 000003
WAITS = 104404           .TRAP = 104400
. ABS.  000000      000
        000120      001
ERRORS DETECTED:  0
FREE CORE:  13385. WORDS
READCW,LP:<SYSYM,READCW.SRC
```

```
$JOB COMPILE INP45
DATE:-18-AUG-77
TIME:-00:10:39
$RUN JOVIAL
```

```
JOVIAL TSS                              *   HARRIS JOVIAL COMPILER
                                            ---VERSION -3A               PAGE    1
      'PROGRAM INP45 $                                                               1
 ''---------------------------------------------------------------------''           2
 ''    INP45 - INPUT PROCESSING FOR THE PDP 11/45 : THIS ROUTINE         ''          2
 ''         PERFORMS THE NECESSARY INITIALIZATION AND THEN ACCEPTS       ''          2
 ''         COMMAND INPUTS FROM EITHER OF THE LSI-11'S OR THE LA36.      ''          2
 ''         THE DESIRED PROCESSING IS CARRIED OUT AND THE RESULTS ARE    ''          2
 ''         RETURNED TO THE CALLING TERMINAL.                            ''          2
 ''    INPUTS:                                                           ''          2
 ''         LRDY = TABLE OF HANDSHAKE FLAGS BETWEEN LSI-11'S AN 45       ''          2
 ''         CBUF = TABLE OF COMMAND INPUT BUFFERS                        ''          2
 ''         DBUF = TABLE OF DATA INPUT BUFFERS                           ''          2
 ''    OUTPUTS:                                                          ''          2
 ''         OUTPUT RESULTS OF QUERY - EVENT OR REPORT TO CALLING LSI     ''          2
 ''---------------------------------------------------------------------''           2
 ''POLL INPUT SOURCES''                                                              2
      DO WHILE GO EQ 0 $                                                             2
           IF LRDY($0$) EQ 0 $                                                       3
                THEN $                                                               4
                CALL SRC45 $                                                         5
                ELSE $                                                               6
           END IF $                                                                  7
           IF LRDY ($1$) EQ 0 $ CALL L1SRC $                                         8
           IF LRDY ($2$) EQ 0 $ CALL L2SRC $                                         10
      END DO $                                                                       12
      ''GO HAS BEEN CANCELLED - EXIT & END''                                         13
      EXIT $                                                                         13
      END SUBROUTINE $                                                               14
      SRC45 SUBROUTINE $                                                             15
     ''THIS SUBROUTINE HANDLES QUERY PROCESSING FROM THE LA36 KEYBOARD''             16
      ''FIRST TRANSFER INPUT COMMAND TO RPN WORK BUFFER ICARD ''                     16
       J = @CBUF($0$) $                                                              16
      TRANC(@CARD,J) $                                                               17
      INSRC = 0 $  ''DESIGNATE INPUT SOURCE''                                        18
      CALL CCNTL  $  ''TRANSLATE & EXECUTE COMMAND''                                 19
      LRDY($0$) = 1  $  ''FINISHED COMMAND''                                         20
      RETURN  $                                                                      21
      END SUBROUTINE $                                                               22
      L1SRC SUBROUTINE $                                                             23
      ''THIS SUBROUTINE HANDLES QUERY PROCESSING FROM LSI NUMBER 1''                 24
      ''TRANSFER INPUT COMMAND TO RPN WORK BUFFER ICARD            ''                24
      J = @CBUF($1$) $                                                               24
      TRANC(@CARD,J) $                                                               25
      INSRC = 1 $  ''DESIGNATE INPUT SOURCE''                                        26
      CALL CCNTL $  ''TRANSLATE & EXECUTE COMMAND''                                  27
      LRDY($1$) = 1 $  ''SIGNAL FINISH''                                             28
      RETURN $                                                                       29
      END SUBROUTINE $                                                               30
      L2SRC SUBROUTINE $                                                             31
      ''THIS SUBROUTINE HANDLES QUERY PROCESSING FROM LSI NUMBER 2''                 32
      ''TRANSFER INPUT COMMAND TO RPN WORK BUFFER ICARD            ''                32
      J = @CBUF($2$) $                                                               32
      TRANC(@CARD,J) $                                                               33
      INSRC = 2  $  ''DESIGNATE INPUT SOURCE''                                       34
      CALL CCNTL $  ''TRANSLATE AND EXECUTE COMMAND''                                35
      LRDY($2$) = 1 $  ''SIGNAL FINISH''                                             36
```

```
      RETURN S                                                  37
      END SUBROUTINE S                                          38
      DIRECT S                                                  39
       MSTK = %5                                                39
       .GLOBL  TRANC                                            39
TRANC: LDF     (MSTK)+,AC0        ;CONVERT ADDR OF CBUF TO      39
       STCFI   AC0,-(MSTK)        ;    FIXED                    39
       MOV     R0,-(SP)           ;SAVE REG.                    39
       MOV     R1,-(SP)           ;                             39
       TST     (MSTK)+                                          39
       MOV     (MSTK)+,R0         ;ADDR OF CBUF IN R0           39
       LDF     (MSTK)+,AC0        ;CONVERT ADDR OF CARD         39
       STCFI   AC0,-(MSTK)        ;   TO FIXED                  39
       TST     (MSTK)+                                          39
       MOV     #80.,R1            ;80 BYTES                     39
1$:                                                             39
       MOV     (R0)+,@(MSTK)+     ;TRANSFER TO CARD             39
       SOB     R1,1$                                            39
       MOV     (SP)+,R1           ;RESTORE REG.                 39
       MOV     (SP)+,R0                                         39
       RTS     PC                                               39
       JOVIAL                                                   39
       END PROGRAM  S                                           39
```

```
    DATA BASE REFERENCE LISTING
S                                   5    13    0    0
CARD                                0     2    1   40
INSRC                             622     9    0   16
GO                                623     9    0   16
LRDY                             1653     9    0   16
CBUF                             1667     9    0   16
J                                   2    13    0   16
COMPOOL SPACE USED =   1108

   0 ERRORS
```

```
$RUN OPT
$RUN MACRO
MACRO V06-04A
*RPN,LP:<SYSYM,PMS,RPNML,MOSML,RPN,COD,OSEND
```

```
SYSMPS  (MULTIPROCESSOR SYSTEM) MACRO V06-04A 18-AUG-77 00:13
TABLE OF CONTENTS

    1-  2  SYSMPS  MULTIPROCESSOR OPERATING SYSTEM
    4-  3  PMMAC PM MACRO DEFINTIONS
    4-125  PMNEQ PROGRAM MODULE NUMBER EQUATES
```

```
                .TITLE  SYSMPS  (MULTIPROCESSOR SYSTEM)
                .SBTTL  SYSMPS  MULTIPROCESSOR OPERATING SYSTEM
```

```
PMMAC PM MACRO DEFINITIONS AND E MACRO V06-04A 18-AUG-77 00:13 PAGE 4
PMNEQ PROGRAM MODULE NUMBER EQUATES

        000002 P45KTR   =002.                    ; RESIDENT KOI FUNCTION FOR 45
        000026 PL1KTR   =022.                    ; RESIDENT KOI FUNCTION FOR L1
        000052 PL2KTR   =042.                    ; RESIDENT KOI FUNCTION FOR L2
        000200 PLOG     =128.                    ;45 SYSTEM ERROR LOGGER
        000201 PCSYLG   = 129.                   ; COMMON SYSTEM ERROR LOGGER
        000202 P45KTO   =130.                    ; OVERLAY  KOI FUNCTION FOR 45
        000203 PPC45    = 131.
        000204 P45SUM   = 132.
        000205 P45CTR   = 133.
        000206 P45S1    = 134.
        000215 PIJOB1   = 141.                   ;----
        000216 PIJOB2   = 142.                   ; INPUT PMS FOR
        000217 PIJOB3   = 143.                   ;    BENCHMARK
        000220 PIJOB4   = 144.                   ;
        000221 PIJOB5   = 145.                   ;---
        000222 PJCTOT   = 146.                   ; OUTPUT JCT TABLE
        000223          PJOVTS = 147.
        000226 PL1KTO   =150.                    ; OVERLAY  KOI FUNCTION FOR L1
        000227 PTCL1 = 151.
        000230 PL1SUM   = 152.
        000231 PL1SA    = 153.
        000241 POJOB1   = 161.                   ;----
        000242 POJOB2   = 162.                   ; OUTPUT PMS FOR
        000243 POJOB3   = 163.                   ;    BENCHMARK
        000244 POJOB4   = 164.                   ;
        000245 POJOB5   = 165.                   ;----
        000252 PL2KTO   =170.                    ; OVERLAY  KOI FUNCTION FOR L2
        000253 PTCL2 = 171.
        000254 PL2SUM   = 172.
        000255 PL2SR    = 173.
```

```
446                                .NLIST  TTM
447                                .NLIST  CND
448         000000                 M.SET=  0
```

```
PMMAC PM MACRO DEFINTIONS AND E MACRO V06-04A 18-AUG-77 00:13 PAGE 6
PMNEQ PROGRAM MODULE NUMBER EQUATES

                        .MACRO  BEGIN ?X,?A1
                        .GLOBL  MSTAK
                X:
                        PSB     A1,24.,FLPT
                A1:     MOV     #MSTAK,MSTK
                        FLTNG
                        .ENDM
                        .MACRO  SETUP X
                        NOP
                        .ENDM
                        .MACRO  STOP
                        NOP
                        .ENDM
                        ; CPOOL ---- FOR ISS UNDER MOS
                        .MACRO  CPOOL A,B,C,?A1
                        .GLOBL  B
                        .CSECT
                        .ENDM
```

```
                                         ;.........                    PDP-11 JOVIAL OPTIMIZER                   ;
                                         ;     'PROGRAM INP45 $                                                  1
000000                                          XDEF  INP45 ,INP45                                               ;
000000                                          MLIB  16                                                         ;
000000                                          CPOOL RPNPOL,COMPOL,03858                                        ;
000000                                          BEGIN INP45                                                      ;
000206                                          SETUP 0                                                          ;
                                         ; ''-----------------------------------------------------------''       2
                                         ; ''   INP45 - INPUT PROCESSING FOR THE PDP 11/45 ; THIS ROUTINE ''    2
                                         ; ''        PERFORMS THE NECESSARY INITIALIZATION AND THEN ACCEPTS ''  2
                                         ; ''        COMMAND INPUTS FROM EITHER OF THE LSI-11'S OR THE LA36. '' 2
                                         ; ''        THE DESIRED PROCESSING IS CARRIED OUT AND THE RESULTS ARE '' 2
                                         ; ''        RETURNED TO THE CALLING TERMINAL.                     ''   2
                                         ; ''   INPUTS:                                                    ''   2
                                         ; ''        LRDY = TABLE OF HANDSHAKE FLAGS BETWEEN LSI-11'S AN 45 ''  2
                                         ; ''        CBUF = TABLE OF COMMAND INPUT BUFFERS                 ''   2
                                         ; ''        DBUF = TABLE OF DATA INPUT BUFFERS                    ''   2
                                         ; ''   OUTPUTS:                                                   ''   2
                                         ; ''        OUTPUT RESULTS OF QUERY - EVENT OR REPORT TO CALLING LSI '' 2
                                         ; ''-----------------------------------------------------------''       2
                                         ; ''POLL INPUT SOURCES''                                               2
                                         ;     DO WHILE GO EQ 0 $                                               2
000210                                   D101:  EQIV  *                                              ;          2
000210                                          STMT       2                                                    ;
000210                                          INTGR                                                           ;
000220  016701  001446G                         MOV   COMPOL+<2*00403.>+000,R1                                  ;
000224  005000                                  CLR   R0                                                        ;
000226                                          FINO  I1001                                          ;          2
                                         ;          IF LRDY($0$) EQ 0 $                                         3
000236                                          STMT       3                                                    ;
000236  005067  000012G                         CLR   XREG00+<2*00005.>+000                                     ;
000242  016704  000012G                         MOV   XREG+<2*5>,R4                                             ;
000246  006304                                  ASL   R4                                                        ;
000250  016401  003526G                         MOV   COMPOL+<2*00939.>(R4),R1                                  ;
000254  005000                                  CLR   R0                                                        ;
000256                                          FINO  I1002                                          ;          3
                                         ;               THEN $                                                 4
                                         ;               CALL SRC45 $                                           5
000266                                          SLJ   SRC45                                          ;          5
                                         ;               ELSE $                                                 6
000272                                          STMT       6                                                    ;
000272                                          GOTO  I9002                                          ;          6
000276                                   I1002: EQIV  *                                              ;          6
                                         ;          END IF $                                                    7
000276                                   I9002: EQIV  *                                              ;          7
                                         ;          IF LRDY ($1$) EQ 0 $ CALL L1SRC $                           8
000276                                          STMT      10                                                    ;
000276  012767  000001  000012G                 MOV   *     1,XREG00+<2*00005.>+000                             ;
000304  016704  000012G                         MOV   XREG+<2*5>,R4                                             ;
000310  006304                                  ASL   R4                                                        ;
000312  016401  003526G                         MOV   COMPOL+<2*00939.>(R4),R1                                  ;
000316  005000                                  CLR   R0                                                        ;
000320                                          FINO  I1004                                          ;          8
000330                                          SLJ   L1SRC                                          ;          9
000334                                   I1004: EQIV  *                                              ;         10
                                         ;          IF LRDY ($2$) EQ 0 $ CALL L2SRC $                          10
000334                                          STMT      12                                                    ;
```

```
     58 000334  012767  000002  000012G              MOV   *      2,XREG00+<2*00005.>+000                          ;
     59 000342  015704  000012G                      MOV   XREG+<2*5>,R4                                            ;
     60 000346  006304                               ASL   R4                                                       ;
     61 000350  016401  003526G                      MOV   COMPOL+<2*00939.>(R4),R1                                 ;
     62 000354  005000                               CLR   R0                                                       ;
     63 000356                                       FIND  I1005                                              ;  10
     64 000366                                       SLJ   L2SRC                                              ;  11
     65 000372                            I1005:     EQIV  *                                                  ;  12
     66                                   ;         END DO $                                                     12
     67 000372                                       STMT     14                                                    ;
     68 000372                                       GOTO  D101                                               ;  12
     69 000376                            I1001:     EQIV  *                                                  ;  13
     70                                             ''GO HAS BEEN CANCELLED - EXIT & END''                         13
     71                                   ;         EXIT $                                                       13
I    72 000376  104411                               EXIT 22                                                    ;
     73                                   ;         END SUBROUTINE $                                             14
     74 000400                                       RETRN                                                      ;
     75                                   ;         SRC45 SUBROUTINE $                                           15
     76 000424                            L03E9:     EQIV  *                                                  ;  15
     77 000424                            SRC45:     EQIV  *                                                  ;  15
     78 000424                                       SETUP 0                                                    ;
     79                                              .GLOBL SRC45                                           ;    15
     80                                   ;        ''THIS SUBROUTINE HANDLES QUERY PROCESSING FROM THE LA36 KEYBOARD''  16
     81                                   ;        ''FIRST TRANSFER INPUT COMMAND TO RPN WORK BUFFER ICARD ''  16
     82                                   ;          J = @CBUF($0$) $                                            16
     83 000426                                       STMT     20                                                    ;
     84 000426  016745  001026                       MOV   IC101 +<2*00000.>+000,-(MSTK)                            ;
     85 000432  005045                               CLR   -(MSTK)                                                  ;
     86 000434                                       PUSHA 0,0,0,0,COMPOL,00951,00,16  ;CBUF                       16
     87 000454                                       POPI  2                          ;J                           16
     88                                   ;         TRANC(@CARD,J) $                                             17
     89 000462                                       FLTNG                                                      ;
     90 000472                                       PUSHA 0,0,0,0,COMPOL,00000,01,40  ;CARD                       17
     91 000512                                       PUSHI 2                          ;J                           17
     92 000520                                       SJ    TRANC                                                    ;
     93                                   ;         INSRC = 0 $  ''DESIGNATE INPUT SOURCE''                      18
     94 000524                                       STMT     22                                                    ;
     95 000524                                       MOVE  0,0,0,0,IC101,0,00,16,0,0,0,0,COMPOL,00402,00,16 ;       ;
     96                                   ;         CALL CCNTL  $  ''TRANSLATE & EXECUTE COMMAND''               19
     97 000550                                       SLJ   CCNTL                                              ;  19
     98                                   ;         LRDY($0$) = 1  $  ''FINISHED COMMAND''                       20
     99 000554                                       STMT     24                                                    ;
    100 000554                                       PUSH  0,0,0,0,IC102,0,00,16                                    ;
    101 000566                                       PUSH  0,0,0,0,IC101,0,00,16                                    ;
    102 000600                                       POPI  5                          ;$                           20
    103 000606                                       POPP  0,0,0,5,COMPOL,00939,00,16  ;LRDY                       20
    104                                   ;         RETURN  $                                                    21
    105 000620                                       RETRN                                                      ;
    106                                   ;         END SUBROUTINE $                                             22
    107 000644                                       RETRN                                                      ;
    108                                   ;         L1SRC SUBROUTINE $                                           23
    109 000670                            L03EA:     EQIV  *                                                  ;  23
    110 000670                            L1SRC:     EQIV  *                                                  ;  23
    111 000670                                       SETUP 0                                                    ;
    112                                              .GLOBL L1SRC                                           ;    23
    113                                   ;        ''THIS SUBROUTINE HANDLES QUERY PROCESSING FROM LST NUMBER 1''  24
    114                                   ;        ''TRANSFER INPUT COMMAND TO RPN WORK BUFFER ICARD          ''  24
```

```
                 ;      J = @CBUF($1$) $                                              24
       000672            STMT      30                                                  ;
       000672            PUSH  0,0,0,0,IC102,0,00,16                                   ;
       000704            PUSHA 0,0,0,0,COMPOL,00951,00,16  ;CBUF                       24
       000724            POPI  2                           ;J                          24
                 ;      TRANC(@CARD,J) $                                              25
       000732            FLTNG                                                         ;
       000742            PUSHA 0,0,0,0,COMPOL,00000,01,40  ;CARD                       25
       000762            PUSHI 2                           ;J                          25
       000770            SJ    TRANC                                                   ;
                 ;      INSRC = 1 $  ''DESIGNATE INPUT SOURCE''                       26
       000774            STMT      32                                                  ;
       000774            MOVE  0,0,0,0,IC102,0,00,16,0,0,0,0,COMPOL,00402,00,16 ;      ;
                 ;      CALL CCNTL $  ''TRANSLATE & EXECUTE COMMAND''                 27
       001020            SLJ   CCNTL                                                 ; 27
                 ;      LRDY($1$) = 1 $  ''SIGNAL FINISH''                            28
       001024            STMT      34                                                  ;
       001024            PUSH  0,0,0,0,IC102,0,00,16                                   ;
       001036            PUSH  0,0,0,0,IC102,0,00,16                                   ;
       001050            POPI  5                           ;$                          28
       001056            POPP  0,0,0,5,COMPOL,00939,00,16  ;LRDY                       28
                 ;      RETURN $                                                      29
       001070            RETRN                                                         ;
                 ;      END SUBROUTINE $                                              30
       001114            RETRN                                                         ;
                 ;      L2SRC SUBROUTINE $                                            31
       001140    L03ER:  EQIV  *                                                     ; 31
       001140    L2SRC:  EQIV  *                                                     ; 31
       001140            SETUP 0                                                       ;
                         .GLOBL L2SRC                                              ;  31
                 ;      ''THIS SUBROUTINE HANDLES QUERY PROCESSING FROM LSI NUMBER 2''  32
                 ;      ''TRANSFER INPUT COMMAND TO PPN WORK BUFFER ICARD             ''  32
                 ;      J = @CBUF($2$) $                                              32
       001142            STMT      40                                                  ;
       001142            PUSH  0,0,0,0,IC103,0,00,16                                   ;
       001154            PUSHA 0,0,0,0,COMPOL,00951,00,16  ;CBUF                       32
       001174            POPI  2                           ;J                          32
                 ;      TRANC(@CARD,J) $                                              33
       001202            FLTNG                                                         ;
       001212            PUSHA 0,0,0,0,COMPOL,00000,01,40  ;CARD                       33
       001232            PUSHI 2                           ;J                          33
       001240            SJ    TRANC                                                   ;
                 ;      INSRC = 2  $  ''DESIGNATE INPUT SOURCE''                      34
       001244            STMT      42                                                  ;
       001244            MOVE  0,0,0,0,IC103,0,00,16,0,0,0,0,COMPOL,00402,00,16 ;      ;
                 ;      CALL CCNTL $  ''TRANSLATE AND EXECUTE COMMAND''               35
       001270            SLJ   CCNTL                                                 ; 35
                 ;      LRDY($2$) = 1 $  ''SIGNAL FINISH''                            36
       001274            STMT      44                                                  ;
       001274            PUSH  0,0,0,0,IC102,0,00,16                                   ;
       001306            PUSH  0,0,0,0,IC103,0,00,16                                   ;
       001320            POPI  5                           ;$                          36
       001326            POPP  0,0,0,5,COMPOL,00939,00,16  ;LRDY                       36
                 ;      RETURN $                                                      37
       001340            RETRN                                                         ;
                 ;      END SUBROUTINE $                                              38
       001364            RETRN                                                         ;
```

```
                                ;       DIRECT $                                          39
                                ;         MSTK = %5                                       39
         000005                           MSTK = %5                                    ;  39
                                ;         .GLOBL  TRANC                                   39
                                          .GLOBL  TRANC                                ;  39
                                ;TRANC:   LDF     (MSTK)+,AC0      ;CONVERT ADDR OF CBUF TO  39
 001410  172425                 TRANC:   LDF     (MSTK)+,AC0      ;CONVERT ADDR OF CBUF TO ;  39
                                ;         STCFI   AC0,-(MSTK)      ;   FIXED                  39
 001412  175445                          STCFI   AC0,-(MSTK)      ;   FIXED              ;  39
                                ;         MOV     R0,-(SP)         ;SAVE REG.                39
 001414  010046                          MOV     R0,-(SP)         ;SAVE REG.             ;  39
                                ;         MOV     R1,-(SP)         ;                         39
 001416  010146                          MOV     R1,-(SP)         ;                      ;  39
                                ;         TST     (MSTK)+                                    39
 001420  005725                          TST     (MSTK)+                                 ;  39
                                ;         MOV     (MSTK)+,R0       ;ADDR OF CBUF IN R0       39
 001422  012500                          MOV     (MSTK)+,R0       ;ADDR OF CBUF IN R0    ;  39
                                ;         LDF     (MSTK)+,AC0      ;CONVERT ADDR OF CARD     39
 001424  172425                          LDF     (MSTK)+,AC0      ;CONVERT ADDR OF CARD  ;  39
                                ;         STCFI   AC0,-(MSTK)      ;   TO FIXED              39
 001426  175445                          STCFI   AC0,-(MSTK)      ;   TO FIXED           ;  39
                                ;         TST     (MSTK)+                                    39
 001430  005725                          TST     (MSTK)+                                 ;  39
                                ;         MOV     #80.,R1          ;80 BYTES                 39
 001432  012701  000120                  MOV     #80.,R1          ;80 BYTES              ;  39
                                ;1$:                                                         39
 001436                         1$:                                                      ;  39
                                ;         MOV     (R0)+,@(MSTK)+   ;TRANSFER TO CARD         39
 001436  012035                          MOV     (R0)+,@(MSTK)+   ;TRANSFER TO CARD      ;  39
                                ;         SOB     R1,1$                                      39
 001440  077102                          SOB     R1,1$                                   ;  39
                                ;         MOV     (SP)+,R1         ;RESTORE REG.             39
 001442  012601                          MOV     (SP)+,R1         ;RESTORE REG.          ;  39
                                ;         MOV     (SP)+,R0                                   39
 001444  012600                          MOV     (SP)+,R0                                ;  39
                                ;         RTS     PC                                         39
 001446  000207                          RTS     PC                                      ;  39
                                ;         JOVIAL                                             39
 001450                                  JOVIAL                                          ;  39
                                ;         END PROGRAM  $                                     39
 001450                         RC101:   DATA    0.000000E 00                               ;
 001460                         IC101:   DATA    00000                                      ;
 001462                         IC102:   DATA    00001                                      ;
 001464                         IC103:   DATA    00002                                      ;
 001466                                  STOP                                               ;
 001470                                  END                                                ;
 001470                                  ENDS                                               ;
```

```
     1          000001                         .END
```

```
PMMAC PX MACRO DEFINTIONS AND F MACRO Y06-04A 18-AUG-77 00:13 PAGE 8-1
SYMBOL TABLE

AC0    =%000000        AC1    =%000001        AC2    =%000002        AC3    =%000003        AC4    =%000004
AC5    =%000005        AC6    =%000006        AC7    =%000007        ARG3   = 000003        B0     = 000001
B1     = 000002        B10    = 002000        B11    = 004000        B12    = 010000        B13    = 020000
B14    = 040000        B15    = 100000        B2     = 000004        B3     = 000010        B4     = 000020
B5     = 000040        B6     = 000100        B7     = 000200        B8     = 000400        B9     = 001000
CCNTL  = ****** G      COMPOL= ****** G       CP     = 000005        DK     = 000001        DUMLOC= 000040
D.BCNT= 000014         D.BUFR= 000012         D.DDF  = 000016        D.DEV  = 000004        D.FL   = 000000
D.FUNC= 000010         D.MCHR= 000005         D.PAGE= 000003         D.PM   = 000002        D.STAT= 000006
D.UNIT= 000005         D101     000210R       EXIT   = 104411        IC101    001460R       IC102    001462R
IC103    001464R       INP45    000000RG      I.ADDR= 000006         I.FL   = 000000        I.TIMR= 000002
I.TYPE= 000004         I1001    000376R       I1002    000276R       I1004    000334R       I1005    000372R
I9002    000276R       KB     = 000000        LP     = 000004        L03EA    000670R       L03EB    001140R
L03E9    000424R       L1SRC    000670RG      L2SRC    001140RG      MSTAK  = ****** G      MSTK   =%000005
MT     = 000002        M.DSA  = 000006        M.FLG  = 000003        M.FLT  = ****** G      M.FWA  = 000004
M.ICBT= 000014         M.ICBW= 000016         M.LGT  = 000010        M.NAM1= 000022         M.NAM2= 000024
M.OPT  = 000012        M.PK   = 000026        M.POPR= ****** G       M.PROC= 000027         M.PUSH= ****** G
M.RUN  = 000001        M.SET  = 000000        M.SUSP= 000002         M.WAIT= 000000         M.WDT  = 000020
PAGE.0= 000000         PAGE.1= 000001         PAGE.2= 000002         PAGE.3= 000003         PAGE.4= 000004
PAGE.5= 000005         PAGE.6= 000006         PCSYLG= 000201         PIJOB1= 000215         PIJOB2= 000216
PIJOB3= 000217         PIJOB4= 000220         PIJOB5= 000221         PJCTOT= 000222         PJOVTS= 000223
PLOG   = 000200        PL1KIO= 000226         PL1KIR= 000026         PL1SA  = 000231        PL1SUM= 000230
PL2KIO= 000252         PL2KIR= 000052         PL2SP  = 000255        PL2SUM= 000254         POJOB1= 000241
POJOB2= 000242         POJOB3= 000243         POJOB4= 000244         POJOB5= 000245         PR     = 177000
PRC45  = 000203        PT     = 000006        PTCL1  = 000227        PTCL2  = 000253        P.AC0  = 000032
P.AC1  = 000042        P.AC2  = 000052        P.AC3  = 000062        P.AC4  = 000072        P.AC5  = 000102
P.PEG  = 000002        P.FLG  = 000026        P.FPS  = 000030        P.PC   = 000020        P.PGF  = 000024
P.PSW  = 000022        P.R0   = 000004        P.R1   = 000006        P.R2   = 000010        P.R3   = 000012
P.R4   = 000014        P.R5   = 000016        P45CTR= 000205         P45KIO= 000202         P45KIR= 000002
P45SUM= 000204         P45S1  = 000206        RC101    001450R       RSPEST= ****** G       R6     =%000006
R7     =%000007        SRC45    000424RG      SWR    = 177570        TASKSW= 104410         TERM   = 104412
TM     = 000003        TRANC    001410RG      WAITS  = 104404        XPEG   = ****** G      XREG00= ****** GX
.TRAP  = 104400
. ABS.   000000     000
         001470     001
ERRORS DETECTED:  1
ERRORS DETECTED:  1
FREE CORE:  8070. WORDS
RPN,LP:<SYSYM,PMS,RPNML,MOSML,RPN.COD,OSEND


FREE CORE:  8070. WORDS
```

```
*
$FINISH
TIME:-00:15:48
```

```
$JOB COMPILE OK SUBS
DATE:-17-AUG-77
TIME:-00:11:01
$RUN PIP
PIP V10-03A
*RPM.SRC/DE

*COMPOL.001/DE

*COMPOL.001<COMPOL.ISS

*RPM.SRC<BI:/FA

$EOD

*
$RUN JOVIAL
```

```
JOVIAL ISS                              *   HARRIS JOVIAL COMPILER
                                            ---VERSION -3A             PAGE    1
     PAGE $                                                                          1
```

```
     POLGN SUBROUTINE   $                                                   2
 ''-------------------------------------------------------------------''    3
 ''  POLISH GENERATOR - ACCEPTS OPERATORS/OPERANDS FROM A SOURCE      ''    3
 ''      STRING AND BUILDS A REVERSE POLISH TARGET STRING             ''    3
 ''  INPUTS ARE:                                                      ''    3
 ''      INTYP = 0 = OPERAND (DATA POINTER)                           ''    3
 ''              1 = OPERATOR (KEYWORD NUMBER)                        ''    3
 ''              2 = OPERAND (CONSTANT POINTER)                       ''    3
 ''              3 = END OF SOURCE STRING                             ''    3
 ''              4 = DATA CONSTANT                                    ''    3
 ''      TTPTR = TAGTABLE POINTER FOR COMPOOL DATA                    ''    3
 ''      TKYNM = KEYWORD NUMBER OF OPERATOR                           ''    3
 ''       CPTR = TAGTABLE POINTER FOR CONSTANT                        ''    3
 ''  OUTPUTS ARE:                                                     ''    3
 ''      POLSTK = A MODIFIED POLISH TARGET STACK (ACCESSED BY PNTRY   ''    3
 ''              (PINDX) )                                            ''    3
 ''       OPSTK = A MODIFIED OPERATOR STACK (ACCESSED BY OPPTR        ''    3
 ''              (OPNDX) )                                            ''    3
 ''-------------------------------------------------------------------''    3
     CASE INTYP $                                                           3
                                                                            4
          OF 0 $           '' 0 MEANS OPERAND ''                            4
          CALL OPRND $                                                      5
                                                                            6
          OF 1 $           '' 1 MEANS OPERATOR ''                           6
          CALL OPRTR $                                                      7
                                                                            8
          OF 2 $            '' OPERAND/CONSTANT ''                          8
          CALL CONST $                                                      9
                                                                           10
          OF 3 $           '' END OF COMMAND ''                            10
          CALL FINSH $                                                     11
                                                                           12
          OF 4 $  '' DATA CONSTANT CHAR ''                                 12
          CALL STRNG $                                                     13
                                                                           14
     END CASE $                                                            14
     RETURN $                                                              15
     END SUBROUTINE $                                                      16
                                                                           17
     OPRND SUBROUTINE $                                                    17
          PNTRY($PINDX$) = TTPTR $                                         18
          PINDX = PINDX + 1 $                                              19
          RETURN $                                                         20
     END SUBROUTINE $                                                      21
                                                                           22
     OPRTR SUBROUTINE $                                                    22
          IF OPNDX EQ 0 $    '' 0 MEANS TOP OF OPERATOR STACK ''           23
               THEN $        '' PUT KEYWORD ON OPSTK ''                    24
               OPPTR($OPNDX$) = TKYNM $                                    25
               OPNDX = OPNDX + 1 $                                         26
               OPPTR($OPNDX$) = ESCHR $                                    27
               OPNDX = OPNDX + 1 $                                         28
               ELSE $        '' COMPARE HIERARCHY ''                       29
               TI = OPPTR($OPNDX - 2$) $                                   30
```

```
        TJ = 0 $
        DO WHILE TJ EQ 0 $
            IF KYHIY(STKYNMS) LQ KYHIY(STTS) $
                THEN $
                OPNDX = OPNDX - 1 $
                PNTRY(SPINDXS) = OPPTR(SOPNDXS) $
                PINDX = PINDX + 1 $
                OPNDX = OPNDX - 1 $
                PNTRY(SPINDXS) = OPPTR(SOPNDXS) $
                PINDX = PINDX + 1 $
                ELSE $
                TJ = 1 $
            END IF $
            IF OPNDX EQ 0 $ TJ = 1 $
            TI = OPPTR(SOPNDX - 2S) $
        END DO $
        OPPTR(SOPNDXS) = TKYNM $
        OPNDX = OPNDX + 1 $
        OPPTR(SOPNDXS) = ESCHR $
        OPNDX = OPNDX + 1 $
    END IF $
    RETURN $
END SUBROUTINE $

CONST SUBROUTINE $
    PNTRY(SPINDXS) = ATCHR $
    PINDX = PINDX + 1 $
    PNTRY(SPINDXS) = CPTR $
    PINDX = PINDX + 1 $
    RETURN $
END SUBROUTINE $

FINSH SUBROUTINE $
    DO UNTIL OPNDX EQ 0 $
        OPNDX = OPNDX - 1 $
        PNTRY(SPINDXS) = OPPTR(SOPNDXS) $
        PINDX = PINDX + 1 $
    END DO $
    RETURN $
END SUBROUTINE $

STRNG SUBROUTINE $
    PNTRY(SPINDXS) = SLASH $
    PINDX = PINDX + 1 $
    PNTRY(SPINDXS) = TTPTR $
    PINDX = PINDX + 1 $
    RETURN $
END SUBROUTINE $
PAGE $
```

```
    KSCAN SUBROUTINE $
''----------------------------------------------------------------------''
''   KEYWORD SCANNER - IDENTIFIES A CHARACTER STRING AS AN OPERATOR     ''
''      OR OPERAND OF A COMMAND                                         ''
''   INPUTS ARE:                                                        ''
''      ICOL = INDEX TO INPUT CHARACTER STRING ICARD                    ''
''     ISLEN = LENGTH OF INPUT CHARACTER STRING (BYTES)                 ''
''     ICARD = TABLE CONTAINING THE COMMAND LINE                        ''
''   OUTPUTS ARE:                                                       ''
''      ICTYP = 0 MEANS NOT A KEYWORD                                   ''
''            = 1 DENOTES KEYWORD                                       ''
''      IKYNM = KEYWORD NUMBER                                          ''
''----------------------------------------------------------------------''
   ICTYP = 0 $      '' NOT A KEYWORD ''
   I = 0 $
   DO UNTIL I GQ NKENT $  '' NKENT IS NUMBER OF KEYWORD ENTRIES ''
        IF  KYLEN($I$) EQ ISLEN $    '' COMPARE LENGTHS ''
             THEN $
             J = 0 $
             L = 0 $
             M = ICOL $
             DO WHILE J LS ISLEN $
                  K = J + KYPTR($I$) $
                  IF KOP($K$) NQ ICARD($M$) $  '' COMPARE CHAR'S ''
                       THEN $
                       L = 1 $    '' MEANS NO MATCH ''
                       ELSE $
                  END IF $
                  J = J + 1 $     '' INCREMENT INDEX ''
                  M = M + 1 $
             END DO $
             IF L EQ 0 $          '' MEANS FOUND ''
                  THEN $
                  ICTYP = 1 $     '' DENOTES KEYWORD ''
                  IKYNM = KYNUM($I$) $  '' RETURN KEYWORD NUMBER ''
                  RETURN $
                  ELSE $
             END IF $
             ELSE $
        END IF $
        I = I + 1 $         '' STEP LOOP INDEX ''
   END DO $
   RETURN $                 '' NOT A KEYWORD ''
   END SUBROUTINE $
   PAGE $
```

```
    TTGEN SUBROUTINE $                                                          113
 ''---------------------------------------------------------------------''     114
 ''   TAGTABLE GENERATOR - IDENTIFIES AN UNDEFINED CHARACTER STRING     ''     114
 ''                        AS A VALID OR INVALID DATA ITEM              ''     114
 ''   INPUTS ARE:                                                       ''     114
 ''      ICARD = TABLE CONTAINING THE COMMAND LINE                      ''     114
 ''      ICOL = INDEX TO INPUT CHARACTER STRING ICARD                   ''     114
 ''      ISLEN = LENGTH OF INPUT CHARACTER STRING (BYTES)               ''     114
 ''   OUTPUTS ARE:                                                      ''     114
 ''      TTPTR = POINTER TO TAGTABLE                                    ''     114
 ''      TTERR = 0 DENOTES INVALID COMPOOL DATA ITEM                    ''     114
 ''              1 ENTRY FOUND IN COMPOOL                               ''     114
 ''---------------------------------------------------------------------''     114
 '' INITIALIZE NTAG AND TAG TO SEND TO SCP ''                                  114
    NTAG = ISLEN $                                                              114
    I = 0 $                                                                     115
    J = ICOL $                                                                  116
    DO UNTIL I GQ ISLEN $                                                       117
       TAG($I$) = ICARD($J$) $                                                  118
       J = J + 1 $                                                              119
       I = I + 1 $                                                              120
    END DO $                                                                    121
    CALL SCP $      '' SEARCH COMPOOL ''                                        123
    IF TERR EQ 0 $                                                              124
       THEN $         '' ADD TAGTABLE ENTRY ''                                  125
       TTPTR = TTPTR + 1 $                                                      126
       TTYP($TTPTR$) = A $                                                      127
       TFB($TTPTR$) = B $                                                       128
       TNB($TTPTR$) = C $                                                       129
       TADDR($TTPTR$) = D $                                                     130
       TTERR = 1 $                                                              131
       ELSE $                                                                   132
       TTERR = 0 $  '' DENOTES INVALID COMPOOL DATA ITEM ''                     133
    END IF $                                                                    134
    RETURN                                                                      135
    END SUBROUTINE $                                                            135
    PAGE $                                                                      136
```

```
    CDCON SUBROUTINE $
''---------------------------------------------------------------''
''   CDCON - CHECKS A STRING TO SEE IF IT IS A DECIMAL STRING AND IF ''
''       SO, CONVERTS IT TO A DECIMAL BINARY NUMBER                ''
''   INPUTS ARE:                                                   ''
''       ICARD = TABLE CONTAINING THE COMMAND LINE                 ''
''        ICOL = INDEX TO INPUT CHARACTER STRING ICARD             ''
''       ISLEN = LENGTH OF INPUT CHARACTER STRING (BYTES)          ''
''   OUTPUTS ARE:                                                  ''
''       CVAL = CONVERTED RESULT - EQUIVALENCED TO TAGTABLE        ''
''        DCR = 0 = NOT A DECIMAL CONSTANT                         ''
''              1 = DENOTES DECIMAL CONSTANT                       ''
''       CPTR = STORE INDEX IN TAGTABLE OF CVAL                    ''
''---------------------------------------------------------------''
    DCR = 1 $
    SUM = 0 $
    I = 0 $
    DO UNTIL I GQ ISLEN $
        J = 0 $
        DO WHILE J LS 10 $
            IF ICARD($ICOL$) EQ IDEC($J$) $
                THEN $
                SUM = SUM * 10 + J $ '' NEW SUM ''
                ICOL = ICOL + 1 $    '' STEP INDEX ''
                J = 10 $     '' END DO ''
                ELSE $
                J = J + 1 $
                IF J EQ 10 $ DCR = 0 $
            END IF $
        END DO $
        I = I + 1 $
    END DO $
    IF DCR EQ 1 $
        THEN $
        CVAL(STTPTR$) = SUM $    '' STORE CONSTANT IN TAGTBL ''
        CPTR = TTPTR $           '' SAVE INDEX ''
        TTPTR = TTPTR + 1 $
        ELSE $
    END IF $
    RETURN $
    END SUBROUTINE $
    END PROGRAM $
```

DATA BASE REFERENCE LISTING

S	5	13	0	0
ICARD	0	14	0	8
ESCHR	74	9	0	8
SLASH	74	9	8	8
ATCHR	75	9	0	8
IDEC	67	14	0	8
ICOL	106	9	0	16
SUM	115	9	0	16
TERR	120	9	0	16
NTAG	122	9	0	16
A	124	9	0	16
B	125	9	0	16
C	126	9	0	16
D	127	9	0	16
TAG	170	14	0	8
KYPTR	226	9	0	16
KYNUM	264	9	0	8
KYHIY	322	9	0	8
KYLEN	360	9	0	8
KOP	416	14	0	8
NKENT	562	9	0	16
ISLEN	563	9	0	16
OPNDX	564	9	0	16
ICTYP	565	9	0	16
PINDX	566	9	0	16
TKYNM	567	9	0	16
TTPTR	570	9	0	16
INTYP	571	9	0	16
DCR	573	9	0	16
CPTR	574	9	0	16
TI	604	9	0	16
TJ	605	9	0	16
TTERR	620	9	0	16
OPPTR	670	14	0	8
PNTRY	740	14	0	8
TTYP	1060	9	0	16
TEB	1104	9	0	16
TNB	1130	9	0	16
TADDR	1154	9	0	16
CVAL	1060	10	0	16
I	1	13	0	16
J	2	13	0	16
K	3	13	0	16
L	4	13	0	16
M	5	13	0	16

COMPOOL SPACE USED = 1138

0 ERRORS

$FINISH
TIME:-00:17:35

MISSION
of
Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C^3) activities, and in the C^3 areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

AMERICAN REVOLUTION BICENTENNIAL 1776-1976